Exploring the Impact of Client Mobility on Decentralized Federated Learning Systems

Santiago de Heredia[1]

Supervisor(s): Jérémie Decouchant[1]**, Bart Cox**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Santiago de Heredia
Final project course: CSE3000 Research Project
Thesis committee: Jérémie Decouchant, Bart Cox, Qing Wang

Abstract

Federated Learning has gained prominence in recent years, in no small part due to its ability to train Machine Learning models with data from users' devices whilst keeping this data private. Decentralized Federated Learning (DFL) is a branch of Federated Learning (FL) that deals with clients directly communicating with each other as opposed to using a central server. Client mobility describes how users' devices move in the real world, and its effects on the learning performance of Hierarchical Federated Learning (HFL) systems have been found to be significant. However, the effects of client mobility on DFL systems have not been explored. In this work, we fill this research gap. First, we develop a model that can describe client mobility in a DFL system. Then, using synthetic datasets, we show that client mobility has a positive impact on learning performance, which we quantify. Moreover, we show that there is a disparity in learning performance between high-mobility and low-mobility clients when using a baseline model aggregation algorithm. To address this disparity, we propose a new mobility-aware model aggregation algorithm. Our experimental results on synthetic datasets show that our solution reduces the disparity in learning performance between high- and low-mobility clients in the scenarios where this disparity is greatest, with no appreciable downsides in global learning performance.

1 Introduction

Federated Learning (FL) is a privacy-preserving machine learning paradigm that permits multiple clients to benefit from a shared model trained from clients' data, sharing model parameters instead of client data [14]. The inception of FL presented a centralized network architecture now called Centralized Federated Learning (CFL), whereby clients send model updates (see Fig. 1c) to a central server, that then performs model aggregation based on these updates [14]. However, since then, a variety of FL architectures have arisen, including Hierarchical Federated Learning (HFL) (see Fig. 2) and Decentralized Federated Learning (DFL) (see Fig. 1d) systems. In HFL, clients send data to a layer of multiple edge servers, which act as intermediaries that subsequently perform model aggregation and forwarding to a central server [13]. In DFL, the need for a centralized controller (server) is foregone completely [9], and instead clients share model parameters with each other using a variety of decentralized communication algorithms [18]. Although less studied, DFL has gained attention for some use-cases due to its inherently decentralized architecture. Certain systems, such as networks of interconnected mobile devices, are particularly well-suited to DFL, with shorter communication ranges conserving bandwidth on constrained devices and improving response times [1]. Other systems crucially benefit from DFL being impervious to Single Point of Failure (SPoF) attacks, and being more resilient to Distributed Denial of Service (DDoS) attacks [7].

Client mobility describes how a client device moves in the real world. The effects of client mobility on HFL systems have been studied in recent years and have been found to be significant [5]. Client mobility in this space has been a subject of investigation due to HFL clients being able to move between different access points between iterations, which affects learning (see Fig. 2). However, as far as we know, there are no works that have studied client mobility in a DFL setting. Additionally, other works have noted that, in DFL, imbalanced parameter exchanges between clients will lead to some models having significantly different parameters than others [6]. Client mobility may make this effect more pronounced, leading to the unfair situation of higher-mobility clients having better models than lower-mobility clients.

Motivated by this research gap and potential unfairness, we analyze the performance of a DFL system with varying levels of client mobility, and propose a mobility-aware model aggregation algorithm to bridge the gap between high- and low-mobility clients. Our contributions can be summarized as follows:

- We develop a theoretical framework to describe client mobility in a DFL system.
- Using synthetic datasets, we show and quantify that increasing the proportion of high-mobility clients has a marked positive impact on global learning performance.
- We identify a disparity in learning performance between high-mobility and low-mobility clients.
- We propose a mobility-aware model aggregation algorithm, designed to bridge the gap between high- and low-mobility clients. Experimental results on synthetic datasets show that this algorithm reduces the disparity in learning performance in scenarios where this disparity is greatest, with no appreciable downsides in global learning performance.

2 Background

McMahan et al. [14] proposed the first and most popular FL algorithm, called federated averaging (FedAvg). At each communication round clients upload their local models to a centralized server, from which a weighted average is computed, with the weights being proportional to the number of samples of the respective client.

Since then, a broad taxonomy of FL systems has emerged [11]. Aside from the different aforementioned

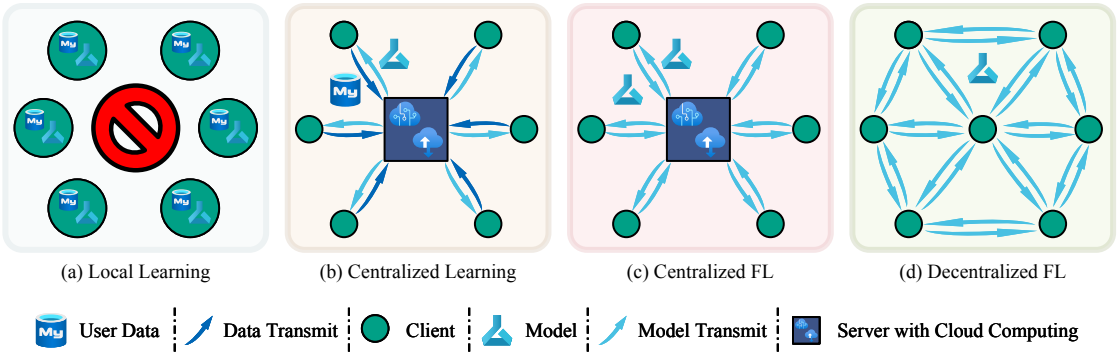

Figure 1: Illustration of different types of learning architectures. We note that (c) corresponds to CFL and HFL, and (d) corresponds to DFL. For HFL, there is an intermediate layer of cloud computing servers, which then aggregate and forward parameters to a centralized server. From [18].

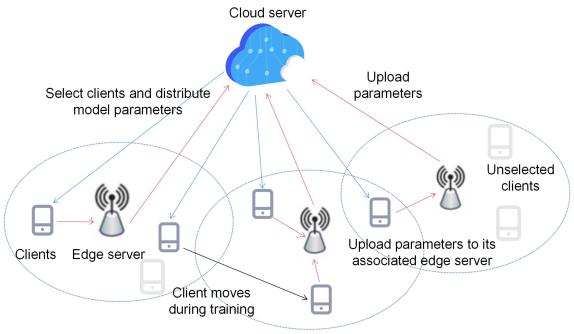

Figure 2: Illustration of a Hierarchical Federated Learning system, with 3 edge servers. Note how clients can move to different edge servers between iterations. From [17].

DFL and HFL communication architectures that were proposed, different *model aggregation techniques* have been proposed [16], many as improvements of FedAvg. These techniques, amongst other things, may involve adjusting the weights, especially since aggregation weights based on the amount of client data (such as in FedAvg) may lead to worse performance [16].

In contrast to CFL, where aggregation is only done in one centralized server, in DFL every client is a *model aggregator*, meaning that the client aggregates the models that it receives with its own model [11]. A number of aggregation algorithms have been proposed, and unlike CFL with FedAvg, there is no aggregation algorithm that is an accepted baseline for DFL [1].

3 Related Work

Mobility in Hierarchical Federated Learning In recent years, the research gap regarding the effects of client mobility in learning performance on HFL systems has gained attention. Feng et al. [5] highlight this gap and develop a theoretical model to characterize client mobility in HFL systems, where clients can move between different access points (APs). Moreover, they propose a mobility-aware cluster FL (MACFL) algorithm, whereby their experiments show that this algorithm improves learning performance compared to baseline methods. Similarly, Feng et al. [4] investigate the effects of mobility on HFL systems through analysis and simulations. However, neither of the mentioned works considers systems other than HFL systems. In contrast, we consider DFL systems.

Bian and Xu [2] address the impact of mobility on the learning performance of asynchronous FL by using mobile relaying between clients, proposing and benchmarking a new FL algorithm for doing so. Similarly, Peng et al. [15] theoretically investigate how mobility affects the performance of FL systems deployed on mobile networks, developing a closed-form expression and finding that client mobility can improve learning performance for the examined mobile network. Nevertheless, we note that both works solely focus on a mobile network setting more reminiscent of HFL, in which clients can interact with intermediary servers (and in the case of Bian and Xu, additionally exchange information with other clients). We deem it important to provide a more synthetic analysis of client mobility, particularly in a fully decentralised system.

Decentralized Federated Learning Frameworks DecentralizePy [3] is an extensible framework for decentralized learning simulations. In the aforementioned work, the authors partly demonstrate the usefulness of their framework by showing how the performance of DFL is affected by network topology. However, the authors do not implement a model for client mobility nor do they conduct experiments to evaluate its effect, which we focus on in this work.

Mobility in Decentralized Federated Learning Gecer et al. [6] give a survey of mobility-related FL solutions and, in doing so, give limitations regarding FL systems for mobility applications. Amongst these, a problem relating to the parameter exchanges in DFL systems is mentioned [6], which highlights how DFL systems may lead to imbalanced parameter exchanges between clients, due to some clients having more neighbours than others. In our work, we recognise that client mobility may exacerbate this effect further (i.e. high-mobility clients will have more neighbours than low-mobility clients), and we explore this possibility further by simulating a variety of DFL systems with varying levels of client mobility.

4 Methodology

Our research methodology involves a mixture of modelling and simulation. We first create a model that is capable of describing client mobility in a DFL system. This is necessary due to the lack of client mobility research on DFL systems. Second, we propose an algorithm to bridge the expected performance gap between high-mobility and low-mobility clients. This gap can be predicted due to the aforementioned imbalanced parameter exchanges in DFL systems [6], which can be further exacerbated due to high-mobility clients having a significantly higher number of neighbours than low-mobility clients. Third, we choose to investigate client mobility by changing the *proportion* (symbolically represented by p) of high-mobility clients within a population sample. This was chosen due to many real-world systems having entities with significantly different mobility distributions. As an example, in a city we would expect many pedestrians (which are lower-mobility), and fewer vehicles (which are higher-mobility). We then generate synthetic data and run experiments for these differing scenarios, recording the learning performance for each algorithm. We later compare, with differing proportions of high-mobility clients, for each algorithm:

- How learning performance is affected
- How the performance gap between high-mobility and low-mobility clients is affected

This methodology allows us to quantify how client mobility affects DFL systems, address potential fairness/performance issues by proposing an appropriate solution, and show that the solution ameliorates these problems.

5 Client Mobility in DFL Systems

In this section, we first explain how clients and their mobility can be modelled, leading to the modelling of a dynamic DFL system. Second, we detail the examined DFL system. Finally, we give the proposed mobility-aware aggregation algorithm, as well as specify the corresponding baseline algorithm.

5.1 Client Mobility Model

World Modeling We model the world as a two-dimensional grid (x, y) with width w and height h. Formally, the world is represented as a continuous space $\mathbb{R}^2$ bounded by $[0, w] \times [0, h]$.

Time For the purpose of simulating clients' movements precisely, time is defined continuously, i.e., $t \in \mathbb{R}$. However, simulations are composed of iterations, which always take 1 unit of time. Formally, for any iteration k, we denote its start time $t_k \in \mathbb{N}$.

Client Mobility Let $\mathcal{U}$ denote the set of all clients. Each client $i \in \mathcal{U}$ at iteration k with start time $t_k \in \mathbb{N}$ is characterized as follows:

- *Movement Direction:* We model the mobility of clients as a random walk [10]. Define a probability vector $\mathbf{q}_i = [u_i, d_i, l_i, r_i]$ for each client i, where u_i, d_i, l_i, and r_i represent the probabilities of moving up, down, left, or right respectively in a time unit. The chosen direction is denoted by $d_i(t_k)$. For this work, we set $u_i = d_i = l_i = r_i = \frac{1}{4}$ for all clients. This is because any statistical bias in direction leads to an unrealistic clustering of clients at the bounds of the grid.
- *Starting Velocity:* Each client i moves with a speed s_i. The initial velocity is given by

  $$v_i(t_k) \in \{(0, -s_i), (0, s_i), (-s_i, 0), (s_i, 0)\}$$

  if $d_i(t_k)$ is up, down, left or right respectively.
- *Position Computation:* We compute the position $p_i \in \mathbb{R}^2$ of client i in iteration k at time $t_k < t + \Delta t \leq t_{k+1}$ by:

  $$p_i(t + \Delta t) = p_i(t) + v_i(t)\Delta t \tag{1}$$

  Where the velocity v_i is computed by:

  $$v_{i,x}(t{+}\Delta t) = \begin{cases} -v_{i,x}(t) & \text{if } p_{i,x}(t){+}v_{i,x}(t)\Delta t{>}w \\ & \text{or } p_{i,x}(t){+}v_{i,x}(t)\Delta t{<}0 \\ v_{i,x}(t) & \text{otherwise} \end{cases}$$
  $$v_{i,y}(t{+}\Delta t) = \begin{cases} -v_{i,y}(t) & \text{if } p_{i,y}(t){+}v_{i,y}(t)\Delta t{>}h \\ & \text{or } p_{i,y}(t){+}v_{i,y}(t)\Delta t{<}0 \\ v_{i,y}(t) & \text{otherwise} \end{cases} \tag{2}$$

  Through this computation, we simulate clients travelling at a constant speed s_i, with their direction d_i reversing if they encounter the grid's boundaries $[0, w] \times [0, h]$. This simulates the clients rebounding off the grid's boundaries, maintaining constant speed but opposite velocity.
- *Neighbours:* The neighbours of a client i at iteration k are defined as the set of all clients $j \in \mathcal{U}$ such that, at any point when moving from time $t - 1$ to t, the

Euclidean distance between client i and client j is less than or equal to the radius R. Formally, the set of neighbours $\mathcal{M}_i(t_k)$ of client i at iteration k, start time t_k is given by

$$\mathcal{M}_i(t_k) = \{j \in \mathcal{U} \mid \exists \tau \in [t_{k-1}, t_k] \text{ s.t. } \| p_i(\tau) - p_j(\tau) \| \leq R\}$$

With the edge-case of $\mathcal{M}_i(t_0) = \{i\}$. This implies that any two clients i and j are neighbours if and only if they are within each other's communication radius between training rounds, simulating direct communication between one-hop neighbours. In our model, every client has the same radius R.

Initial Parameters The initial parameters for each experiment are distributed as follows:

- *Initial client positions:* The initial position of each client $i \in \mathcal{U}$ $p_i(0) = (x_i(0), y_i(0))$ is determined by i.i.d uniform random variables:

$$x_i(0) \sim \text{Uniform}(0, w), \quad y_i(0) \sim \text{Uniform}(0, h).$$

- *Client speeds:* Define the following variables:

$$s_{\text{max}} = \text{maximum velocity for low-mobility clients}$$
$$\beta = \text{scaling factor for high-mobility clients}$$

For low-mobility clients, we have that:

$$s_i \sim \text{Uniform}(0, s_{\text{max}})$$

Otherwise, for high-mobility clients, we have that:

$$s_i \sim \text{Uniform}(s_{\text{max}} \times \beta, 2 \times s_{\text{max}} \times \beta)$$

Where $\beta > 1$ ensures that high-mobility clients have a higher velocity range.

These distributions guarantee that all high-mobility clients will be faster than all low-mobility clients by a factor of at least β.

5.2 System Setup

DFL encompasses a wide range of categories and types [18], making it essential for us to focus on a particular system for our analysis. For the optimizer, we selected a basic stochastic gradient descent (SGD) optimizer with logarithmic loss. For the communication algorithm between clients and model aggregation, we chose decentralized parallel stochastic gradient descent (D-PSGD) [12]. D-PSGD is a decentralized learning algorithm in which clients avoid needing a central server by relying on given communication topologies. Models from neighbouring clients for a given client i at iteration k are averaged following Alg. 1, where for each neighbouring client j the importance of its model x_j^k is determined by its corresponding weight W_{ij}. A variety of weighting schemes W can be used - we now present the baseline for benchmarking and later the proposed solution.

5.3 Baseline Model Aggregation Algorithm

For the baseline comparison, we use *plain* model averaging, that is, every neighbouring model has equal importance in the averaging. This leads to the same weight matrix W that is presented in the original D-PSGD publication [12]. Concretely, for a given client i at iteration k, start time t_k, for any neighbour $j \in \mathcal{M}_i(t_k)$, we define its corresponding model weight W_{ij} as

$$W_{ij} = \frac{1}{N}$$

where $N = \|\mathcal{M}_i(t_k)\|$, i.e. the number of neighbours of i at iteration k.

5.4 Mobility-Aware Model Aggregation Algorithm

Here we propose a solution to the presumed performance disparity between low- and high-mobility clients, due to high-mobility clients on average having a significantly higher number of neighbours over multiple rounds, thus having an imbalanced parameter exchange and better learning performance. The principal idea is to leverage this foresight and the additional assumption that clients know the velocities of their neighbours to weigh their neighbour's models proportionally to their velocity.

A naive approach would be to normalise the neighbours' speeds relative to the other neighbours and use these for the aggregation weights. Concretely, for a given client i at iteration k, start time t_k, for any neighbour $j \in \mathcal{M}_i(t_k)$, we define its corresponding normalised speed X_{ij} as

$$X_{ij} = \frac{s_j}{\sum_{x \in \mathcal{M}_i(t_k)} s_x}$$

Preliminary experiments showed that using X_{ij} as the aggregation weight would lead to too much bias towards the neighbour's speed. To control for this, we introduce a ratio hyperparameter α to calculate the final aggregation weights W_{ij} as

$$W_{ij} = \frac{1}{N} + \alpha(X_{ij} - \frac{1}{N})$$

where $N = \|\mathcal{M}_i(t_k)\|$, i.e. the number of neighbours of i at iteration k. We note that at $\alpha = 0.0$, the algorithm devolves into the plain averaging baseline of Sec. 5.3, whereas at $\alpha = 1.0$ we have naive $W_{ij} = X_{ij}$.

6 Evaluation

In this section, we first demonstrate the performance effects of client mobility, as well as the performance gap between low- and high-mobility clients, on the previously-described DFL system in Sec. 5.2. Moreover, we compare and contrast these aforementioned properties with the proposed model-averaging algorithm in

Algorithm 1 Our adaptation of D-PSGD with dynamic topologies for client i (logical view). Adapted from [12]

Require: Initialize local model $\{x_0^i\}$ with the initialization, learning rate γ, batch size M, and total number of iterations K.

1: **for** $k = 0, 1, \ldots, K-1$ **do**
2: Randomly sample a batch $\xi_k := (\xi_1, \xi_2, \ldots, \xi_M)$ from local data of client i.
3: Compute the stochastic gradient locally $g_k(\hat{x}_k^i; \xi_k^i) := \sum_{j=1}^{M} \nabla F(\hat{x}_k^i; \xi_{k,j}^i)$.
4: Partially update the local model $x_{k+1/2}^i \leftarrow x_k^{i_k} - \gamma g_k(\hat{x}_k^{i_k}; \xi_k^{i_k})$
5: Contact its neighbours $j \in \mathcal{M}_i(t_k)$ and receive their partial models $x_{k+1/2}^j$.
6: Average local model with neighbours' models by $x_{k+1}^i \leftarrow \sum_{j \in \mathcal{M}_i(t_k)} W_{ij} x_{k+1/2}^j$
7: **end for**
8: Output the average of the models on all workers for inference.

Sec. 5.4. We give an in-depth analysis of results for systems with non-IID data partitioning in Sec. 6.2 through Sec. 6.4, as well as a summary for systems IID data partitioning in Sec. 6.5.

6.1 Experimental Setup

The experiments are run on one hyperthreading-enabled machine on 2 Intel Xeon Gold 6253CL @ 3.10GHz, having access to a total of 30 cores and 240GB of RAM. For the dataset, we use the CIFAR-10 dataset [8] with non-IID data partitioning. The experiments are run with a total of $N = 48$ clients. Each experiment is run 3 times, each with a new randomised seed, from which we take and present the average.

To simulate these DFL systems, we use an extended version of DecentralizePy [3], an easily extensible framework for decentralized ML. The source code for the extended version used in this work, along with random seeds and extra configurations used for these experiments, can be found in a publicly available online repository[2].

6.2 Performance Effects of Increasing High-Mobility Clients

The results of these experiments are summarised in Tab. 1.

Proportion (p)	Accuracy improvement relative to $p = 0$ (%)
5%	+2.73
20%	+4.64
40%	+6.51
60%	+6.71
80%	+8.56
100%	+8.97

Table 1: Average final test accuracy difference from $p = 0\%$ for varying proportions of high-mobility clients p

[2] https://github.com/asturnox/decentralizepy

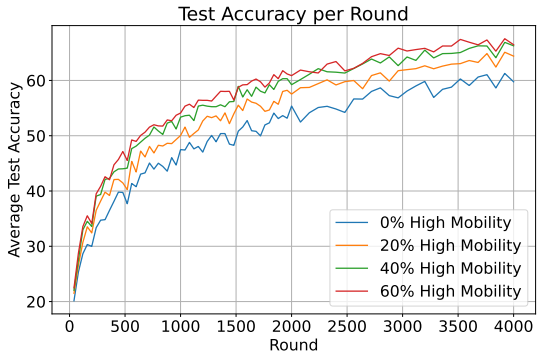

Figure 3: Test accuracy per round for systems with 0%, 20%, 40% and 60% high-mobility clients.

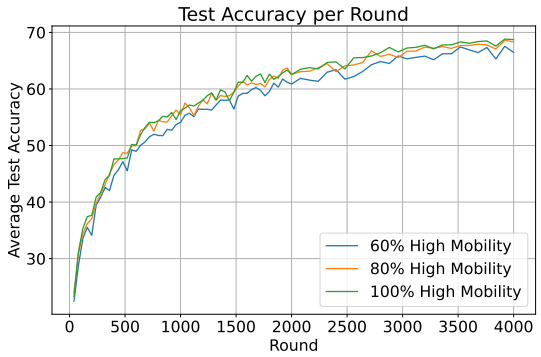

Figure 4: Test accuracy per round for systems with $p = 60\%$, 80% and 100%. Returns in increased test accuracy are negligible beyond $p = 60\%$.

The results of these experiments show a marked improvement in test accuracy when increasing p from 0% to 20%, 40%, 60% (see Fig. 3). We observe that the biggest change occurs from 0% to 20%, after which returns appear to be diminishing. This is confirmed by comparing performance with systems with $p = 60\%$, 80% and 100%

Proportion (p)	Average test accuracy advantage (%)
5%	+4.81
20%	+4.13
40%	+2.53
60%	+2.04
80%	+0.90

Table 2: Average test accuracy advantage throughout experiment between high-mobility and low-mobility clients for varying levels of p.

(see Fig. 4), in which differences appear to be less pronounced, especially between $p = 80\%$ and $p = 100\%$.

These diminishing returns are explained by high values of p leading to low-mobility clients have a similar amount of neighbours than their high-mobility counterparts, as most of their neighbours become high-mobility clients. This means that further replacement of low-mobility clients with high-mobility clients does not lead to much more parameter exchanges. A comparison between the number of neighbours of high- and low-mobility clients can be found in App. B.

We also observe that having a relatively small p has an outsized impact on learning performance compared to $p = 0\%$, noting a 2.73% improvement by just increasing p to 5%.

6.3 Performance Gap Between High-Mobility and Low-Mobility Clients

The difference in learning performance between high- and low-mobility clients, for varying levels of p, is summarised in Tab. 2. It should be noted that due to the variability of test accuracies at the end of each experiment for high-mobility clients (due to some scenarios having a limited amount of these clients), the average test accuracy difference throughout the experiment is given instead.

The results of these experiments show that high-mobility clients have a clear learning advantage over low-mobility clients. In particular, in Fig. 5 we observe that, in environments with a low proportion of high-mobility clients (in this case $p = 5\%$), high-mobility clients generally have a significantly higher test accuracy throughout. This confirms our presumption that high-mobility clients would have better learning performance due to the increased amount of neighbours throughout iterations leading to more voluminous and varied parameter exchanges.

Interestingly, we also note that increasing p significantly past 20% (see Fig. 6) closes the gap between high-mobility and low-mobility clients, with low-mobility clients improving their learning performance. This can be explained by low-mobility clients communicating with more high-mobility clients (due to the latter becoming much more common), thus resulting in a significant increase in parameter exchanges for lower-mobility clients, thus providing more varied parameters for aggregation and closing this gap with high-mobility clients.

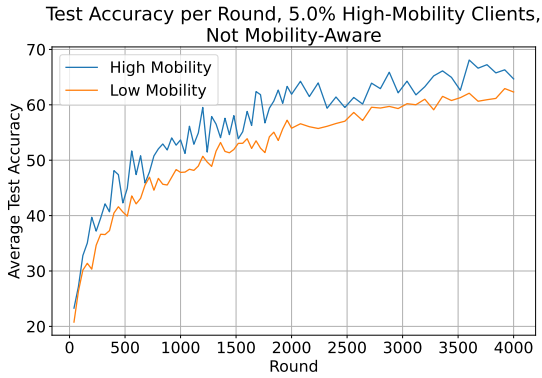

Figure 5: A comparison of high-mobility vs low-mobility clients in a system with $p = 5\%$. There is a significant difference in learning performance between high- and low-mobility clients.

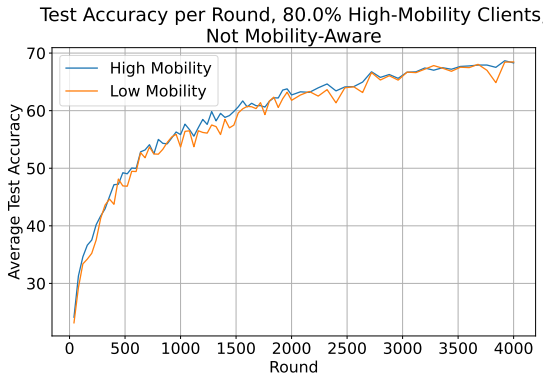

Figure 6: A comparison of High-Mobility vs Low-Mobility clients in a system with $p = 80\%$. There is a negligible difference in learning performance between high- and low-mobility clients.

6.4 Comparison with Mobility-Aware Aggregation Algorithm

Given the results of the plain-averaging algorithm baseline, we now compare these results to using the aforementioned mobility-aware aggregation algorithm in Sec. 5.4. We first compare the performance difference between low- and high-mobility clients, followed by a comparison in global learning performance.

Difference Between High-Mobility and Low-Mobility Clients

In environments with a low proportion of high-mobility clients ($p = 5\%$ and $p = 20\%$), for all configurations of α, we show (see Tab. 3) a less exaggerated performance gap between high- and low-mobility clients. In partic-

Table 3: Average test accuracy difference (%) between HM and LM clients throughout experiments, different aggregation methods.

	$p = 0.05$	$p = 0.2$	$p = 0.4$	$p = 0.6$	$p = 0.8$
Plain Averaging	+4.81	+4.13	+2.53	+2.04	+0.90
Mobility-Aware ($\alpha = 0.1$)	+4.66	+3.79	+2.84	+1.90	+1.27
Mobility-Aware ($\alpha = 0.2$)	+3.35	+3.70	+3.00	+2.35	+1.20
Mobility-Aware ($\alpha = 0.3$)	+4.26	+3.92	+2.81	+1.97	+1.24
Mobility-Aware ($\alpha = 0.4$)	+3.16	+3.17	+2.64	+2.28	+1.19
Mobility-Aware ($\alpha = 1.0$)	+3.76	+3.45	+3.47	+3.44	+2.89

ular, we note that for this particular system, mobility-aware aggregation with $\alpha = 0.4$ seems to be well-suited, with absolute differences in test accuracy being reduced by more than 1.5% in the scenario where this difference is the highest (i.e. $p = 5\%$), by around 1.0% in the scenario with the second-highest gap ($p = 20\%$), and similar disparities to the baseline in higher-mobility scenarios (p values of 40%, 80%, 100%).

Nevertheless, we also note in Tab. 3 that the proposed solution is sensitive to the hyperparameter α. While all values of α lead to lower or similar differences between clients for values of $p = 5\%$ and $p = 20\%$ than the baseline, increasing p sometimes leads to the opposite intended effect - with the baseline leading to a significantly lower difference in the naive case of $\alpha = 1.0$.

This trend reversal could be explained by the fact that, given a critical proportion of high-mobility clients, low- and high-mobility clients have a similar amount of neighbours $\|\mathcal{M}_i(t_k)\|$ and thus a similar variety of parameter exchanges. Given this scenario, the heuristic of biasing towards higher-mobility clients fails (as both client classes are now similar), and a plain averaging algorithm is indeed fairer and better performing.

Overall Learning Performance Comparison

For mobility-aware aggregation with α values of 0.1, 0.2, 0.3, 0.4, we see a negligible difference in learning performance compared to plain averaging throughout values of p. Concretely, we see the largest absolute difference when $\alpha = 0.1$ and $p = 0.2$, in which the baseline has a 1.53% higher final test accuracy. Moreover, we note that there is no identifiable trend in learning differences - increasing p for an equal value of α may lead to both negligibly positive or negative improvements. A complete table of results can be found in App. A.

A possible explanation may be that the values of α may be too small, leading to the weights being too similar to those of the baseline, thus leading to a reduced difference. However, this is disputed by the fact that we did see consistently lower differences between high- and low-mobility clients (in lower-mobility scenarios) in the previous section. Another possible explanation may be that despite biasing weights towards clients with more complete models (i.e., higher mobility), we may be losing too much knowledge by ignoring lower-mobility clients. We may thus have two opposing effects with no truly beneficial compromise for learning performance: biasing towards good models (improving accuracy), but ignoring worse models (deteriorating accuracy).

This latter explanation seems to be verified by the naive case of $\alpha = 1.0$, where we observe (see Tab. 4) a clear decrease in performance compared to using plain averaging. Interestingly, we note that this difference is highest at around $p = 20\%$, after which this difference tends to 0.

Expanding on the previous explanation, this decreased performance in environments with low p stems from small differences in velocity affecting aggregation weights substantially to no major benefit - as an example, a low-mobility client with a movement speed s close to 0 may still encounter many neighbours and have useful parameters, but its model will be essentially discarded due to its low velocity. This lack of attention to extremely low-mobility clients is reminiscent of problems that can arise in FedAvg from clients that have low amounts of data, due to model weights based on the amount of data each client possesses [16].

6.5 Summary of IID Results

Here we summarise the analogous results to the previous sections, but with IID data partitioning.

- *Performance Effects of Increasing HM Clients:* In general, we see significantly lower learning performance improvements when increasing p (see Tab. 5) than in the non-IID scenario. However, similarly to the non-IID scenario, we still observe diminishing returns in learning performance, with most of the improvements stemming from $p = 0\%$ to $p = 60\%$.

Proportion (p)	Accuracy difference with naive mobility-aware aggregation (%)
0%	-2.28
5%	-4.20
20%	-5.11
40%	-4.51
60%	-2.03
80%	-1.20
100%	-0.56

Table 4: Average final test accuracy difference between naive ($\alpha = 1.0$) mobility-aware and plain averaging for varying values of p

Proportion (p)	Average test accuracy improvement (%)
5%	+0.07
20%	+1.12
40%	+1.08
60%	+1.43
80%	+1.90
100%	+2.11

Table 5: Average final test accuracy difference from $p = 0\%$ for varying values of p, IID data distribution

Proportion (p)	Average test accuracy advantage (%)
5%	+1.57
20%	+1.21
40%	+0.94
60%	+0.85
80%	+0.60

Table 6: Average test accuracy difference throughout experiments between high-mobility and low-mobility clients for varying levels of p, IID data distribution

- *Performance Gap Between HM and LM Clients:* We generally see a smaller advantage between high- and low-mobility clients (see Tab. 6) compared to the non-IID scenario. Nevertheless, we observe that this advantage still closes as p increases, which reflects the non-IID scenario.
- *Performance of Mobility-Aware Aggregation Algorithm:* Similarly to the non-IID results, we see no significant differences in learning performance between both algorithms. A full table of results can be found in App A.

These generally less pronounced results come entirely due to the dataset being IID. When the dataset is non-IID, increasing the number of higher-mobility clients leads to more parameter exchanges for the entire system, thus increasing the likelihood that representative samples are exchanged at each model aggregation step [14]. However, this does not apply to IID datasets, thus leading to much more modest learning performance improvements. The same logic can be applied to the lowered performance gap between high- and low-mobility clients: high-mobility clients are not exaggeratedly more likely to receive more representative samples than low-mobility clients, hence we can expect a lower performance gap.

7 Responsible Research

The author declares that they have no known competing financial interests or personal relationships that could have appeared to influence this work. Moreover, to help ensure reproducibility, all data, code and analyses used in this work have been publicly published and linked. Finally, resources that were used to produce this work, including DecentralizedPy and CIFAR-10, have been appropriately referenced.

8 Conclusions and Future Work

In conclusion, this work investigated the effects of client mobility on Decentralized Federated Learning systems, identifying a performance gap between high- and low-mobility clients and proposing a new aggregation algorithm to close this gap. We showed that increasing the proportion p of high-mobility clients results in substantial increases in test accuracy until $p = 80\%$, with returns diminishing past $p = 20\%$. We showed that, despite a higher test accuracy between high- and low-mobility clients in lower-mobility scenarios ($p = 5\%$ and $p = 20\%$), this difference can be closed by using a mobility-aware aggregation algorithm, without impacting overall learning performance. Finally, we demonstrated that these results are far more exaggerated in non-IID scenarios, whereas IID scenarios show fewer performance improvements as p increases and smaller learning performance disparities between high- and low-mobility clients.

Due to the synthetic nature of the generated mobility dataset, the authors recognise that the results of this work might diverge from real-world systems. As such, an interesting direction for future work would be to use real-life mobility traces. Moreover, due to the broad taxonomy of Decentralized Federated Learning systems [11], investigating a Decentralized Federated Learning system with a different communication protocol, aggregation paradigm or iteration order is another interesting direction for future work. Finally, automating the α hyperparameter in the mobility-aware aggregation algorithm could be a valuable future work.

A Learning Performance Comparison

Here, we present a direct comparison in learning performance between using the baseline plain averaging algorithm and mobility-aware aggregation for different values of its hyperparameter α. As explained in the evaluation, neither for the IID nor the non-IID cases do we see a significant difference in learning performance, except for the naive mobility-aware aggregation with $\alpha = 1.0$. The non-IID comparison can be found in Tab. 7, and IID results can similarly be found in Tab. 8

A.1 Non-IID Results

Table 7: Average final test accuracy difference (%) between mobility-aware aggregation and plain-aggregation, for varying values of p and α, non-IID data distribution.

	$p = 0.0$	$p = 0.05$	$p = 0.2$	$p = 0.4$	$p = 0.6$	$p = 0.8$	$p = 1.0$
Mobility-Aware ($\alpha = 0.1$)	+0.02	+0.85	-1.53	+0.42	+1.33	-0.05	-0.70
Mobility-Aware ($\alpha = 0.2$)	+0.29	-0.37	+0.16	-0.07	+0.91	+0.01	-0.13
Mobility-Aware ($\alpha = 0.3$)	+0.91	-1.30	+0.05	-0.63	+0.53	+0.13	-0.30
Mobility-Aware ($\alpha = 0.4$)	+0.98	-1.40	-0.43	-1.05	+0.13	-1.17	-0.30
Mobility-Aware ($\alpha = 1.0$)	-2.28	-4.20	-5.11	-4.51	-2.03	-1.20	-0.56

A.2 IID Results

Table 8: Average final test accuracy difference (%) between mobility-aware aggregation and plain-aggregation, for varying values of p and α, IID data distribution.

	$p = 0.0$	$p = 0.05$	$p = 0.2$	$p = 0.4$	$p = 0.6$	$p = 0.8$	$p = 1.0$
Mobility-Aware ($\alpha = 0.1$)	-0.23	+0.39	-0.53	+0.18	+0.18	-0.09	-0.05
Mobility-Aware ($\alpha = 0.2$)	+0.17	+0.23	-0.42	-0.01	+0.06	+0.02	-0.09
Mobility-Aware ($\alpha = 0.3$)	-0.34	-0.05	-1.03	+0.01	-0.03	-0.03	-0.03
Mobility-Aware ($\alpha = 0.4$)	-0.90	+0.12	-1.02	-0.14	-0.13	-0.01	-0.05
Mobility-Aware ($\alpha = 1.0$)	-1.73	-2.26	-4.37	-1.98	-1.98	-0.76	-0.24

B Number of Neighbours Comparison

Here we show how varying p affects the number of neighbours for both high- and low-mobility clients. Fig. 7 shows a significantly wider gap in neighbours between an average high- and low-mobility node with $p = 5\%$ compared to Fig. 8, where this difference is smaller due to the higher value of $p = 80\%$.

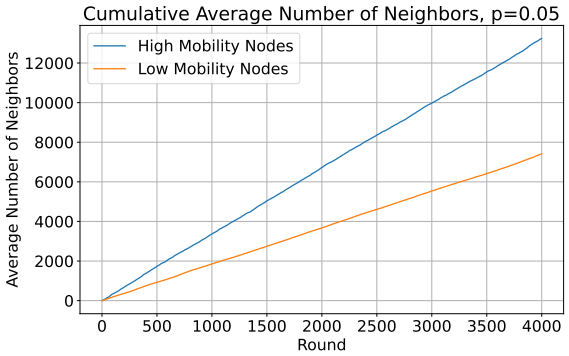

Figure 7: Cumulative number of neighbours on average for high-mobility and low-mobility nodes, $p = 5\%$

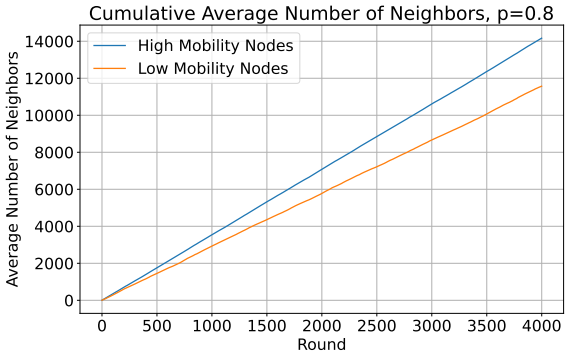

Figure 8: Cumulative number of neighbours on average for high-mobility and low-mobility nodes, $p = 80\%$

References

[1] BELTRÁN, E. T. M., PÉREZ, M. Q., SÁNCHEZ, P. M. S., BERNAL, S. L., BOVET, G., PÉREZ, M. G., PÉREZ, G. M., AND CELDRÁN, A. H. Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges. *IEEE Communications Surveys & Tutorials* (2023).

[2] BIAN, J., AND XU, J. Accelerating asynchronous federated learning convergence via opportunistic mobile relaying. *IEEE Transactions on Vehicular Technology* (2024), 1–13.

[3] DHASADE, A., KERMARREC, A.-M., PIRES, R., SHARMA, R., AND VUJASINOVIC, M. Decentralized learning made easy with decentralizepy. In *Proceedings of the 3rd Workshop on Machine Learning and Systems* (New York, NY, USA, 2023), EuroMLSys '23, Association for Computing Machinery, p. 34–41.

[4] FENG, C., YANG, H. H., HU, D., QUEK, T. Q., ZHAO, Z., AND MIN, G. Federated learning with user mobility in hierarchical wireless networks. In *2021 IEEE Global Communications Conference (GLOBECOM)* (2021), pp. 01–06.

[5] FENG, C., YANG, H. H., HU, D., ZHAO, Z., QUEK, T. Q. S., AND MIN, G. Mobility-aware cluster federated learning in hierarchical wireless networks. *IEEE Transactions on Wireless Communications 21*, 10 (2022), 8441–8458.

[6] GECER, M., AND GARBINATO, B. Federated learning for mobility applications. *ACM Comput. Surv. 56*, 5 (jan 2024).

[7] HALLAJI, E., RAZAVI-FAR, R., SAIF, M., WANG, B., AND YANG, Q. Decentralized federated learning: A survey on security and privacy. *IEEE Transactions on Big Data* (2024).

[8] KRIZHEVSKY, A., HINTON, G., ET AL. Learning multiple layers of features from tiny images.

[9] LALITHA, A., KILINC, O. C., JAVIDI, T., AND KOUSHANFAR, F. Peer-to-peer federated learning on graphs. *arXiv preprint arXiv:1901.11173* (2019).

[10] LAWLER, G. F., AND LIMIC, V. *Random walk: a modern introduction*, vol. 123. Cambridge University Press, 2010.

[11] LI, Q., WEN, Z., WU, Z., HU, S., WANG, N., LI, Y., LIU, X., AND HE, B. A survey on federated learning systems: Vision, hype and reality for data privacy and protection. *IEEE Transactions on Knowledge and Data Engineering 35*, 4 (2021), 3347–3366.

[12] LIAN, X., ZHANG, C., ZHANG, H., HSIEH, C.-J., ZHANG, W., AND LIU, J. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. *Advances in neural information processing systems 30* (2017).

[13] LIU, L., ZHANG, J., SONG, S., AND LETAIEF, K. B. Client-edge-cloud hierarchical federated learning. In *ICC 2020-2020 IEEE international conference on communications (ICC)* (2020), IEEE, pp. 1–6.

[14] MCMAHAN, B., MOORE, E., RAMAGE, D., HAMPSON, S., AND Y ARCAS, B. A.

Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics* (2017), PMLR, pp. 1273–1282.

[15] PENG, Y., TANG, X., ZHOU, Y., HOU, Y., LI, J., QI, Y., LIU, L., AND LIN, H. How to tame mobility in federated learning over mobile networks? *IEEE Transactions on Wireless Communications 22*, 12 (2023), 9640–9657.

[16] QI, P., CHIARO, D., GUZZO, A., IANNI, M., FORTINO, G., AND PICCIALLI, F. Model aggregation techniques in federated learning: A comprehensive survey. *Future Generation Computer Systems* (2023).

[17] YANG, J., ZHOU, Y., WEN, W., ZHOU, J., AND ZHANG, Q. Asynchronous hierarchical federated learning based on bandwidth allocation and client scheduling. *Applied Sciences 13*, 20 (2023).

[18] YUAN, L., SUN, L., YU, P. S., AND WANG, Z. Decentralized federated learning: A survey and perspective. *arXiv preprint arXiv:2306.01603* (2023).