ns3-fl: Simulating Federated Learning with ns-3

Emily Ekaireb
University of California San Diego
La Jolla, California, United States
eekaireb@ucsd.edu

Xiaofan Yu
University of California San Diego
La Jolla, California, United States
x1yu@ucsd.edu

Kazim Ergun
University of California San Diego
La Jolla, California, United States
kergun@ucsd.edu

Quanling Zhao
University of California San Diego
La Jolla, California, United States
quzhao@ucsd.edu

Kai Lee
University of California San Diego
La Jolla, California, United States
kal030@ucsd.edu

Muhammad Huzaifa
University of California San Diego
La Jolla, California, United States
mhuzaifa@ucsd.edu

Tajana Rosing
University of California San Diego
La Jolla, California, United States
tajana@ucsd.edu

ABSTRACT

In recent years, there has been a spike in interest in the field of federated learning (FL). As a result, an increasing number of federated learning algorithms have been developed. Large-scale deployments to validate these algorithms are often not feasible, resulting in a need for simulation tools which closely emulate real deployment conditions. Existing federated learning simulators lack complex network settings, and instead focus on data and algorithmic development. ns-3 is a discrete event network simulator, which has a plethora of models to represent network components and can simulate complex networking scenarios. In this paper, we present ns3-fl, which is a tool that connects an existing FL simulator, flsim, with ns-3 to produce a federated learning simulator that considers data, algorithm, and network. We first discuss the learning, network and power models used to develop our tool. We then present an overview of our implementation, including the Client/Server ns-3 applications and interprocess communication protocols. A real Raspberry Pi-based deployment is setup to validate our tool. Finally, we perform a simulation emulating FL training on 40 clients throughout the UCSD campus and analyze the performance of our tool, in terms of real clock execution time for various FL rounds.

CCS CONCEPTS

• **Networks** → **Network simulations**; • **Computing methodologies** → *Machine learning*.

KEYWORDS

federated learning, ns-3, network simulation

ACM Reference Format:
Emily Ekaireb, Xiaofan Yu, Kazim Ergun, Quanling Zhao, Kai Lee, Muhammad Huzaifa, and Tajana Rosing. 2022. ns3-fl: Simulating Federated Learning with ns-3. In *Proceedings of the WNS3 2022 (WNS3 2022), June 22–23, 2022, Virtual Event, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3532577.3532591

1 INTRODUCTION

With the growth of computational power among devices in distributed networks, such as smartphones, smartwatches, and autonomous vehicles, interest in federated learning (FL) has been increasing among researchers. The number of publications that include "federated learning" in their title has increased tenfold from 2019 to 2021[1]. In traditional machine learning, clients send raw data to the server which is used to train the global model. In contrary, FL has clients train on local datasets, and only send the learned model to the server for aggregation and global updates. Such learning paradigm enables distributed training on local dataset while preserves user privacy [15].

The practical performance of FL depends on an interplay between data, algorithm, and network. However, *none of the existing simulators consider all of these factors.* The existing simulators support training over various datasets, and focus on propelling forward algorithmic development, but do not take into consideration complex network behaviors. They assume ideal networks [21] or have static behaviors to represent network features such as client drop out and latency of communication [17, 22]. Despite this, the convergence of training is constrained by power budgets and connection quality of clients, as well as potential over-saturation over network channels [18]. Thus, the network plays a crucial role in convergence but usually is ignored in existing simulation tools. Because many of the open issues in FL are motivated by real-world FL constraints and settings, overlooking the network behaviors could make a system appear to work well in simulation but lack reproducible results in practice [13]. In our work, we focus on creating a FL simulator which takes into account the **data, algorithm, and network**.

The contribution of this paper is as follows:

[1]This figure was determined through Google Scholar results for key word "federated learning".

(1) We create a simulating framework ns3-fl for researchers to validate their FL algorithms under practical networking conditions. We implement the framework using a PyTorch-based FL simulator flsim [22] and ns-3. The data, algorithm and network models in ns3-fl can be easily extended by the user. Our source code is accessible online at https://github.com/eekaireb/ns3-fl.
(2) We validate the framework by comparing the convergence time and energy consumption results between ns3-fl and a real deployment of Raspberry Pis. Our results demonstrate at most 1.63% difference in the accuracy, at most 0.95% difference in the convergence times and a maximal 10.54% difference in the energy consumption.
(3) We perform a simulation emulating 40 Raspberry Pis on the UCSD campus to show how ns3-fl can be used to support decision design for FL deployments, including data distribution, network quality, and algorithm configuration.

The rest of the paper is organized as follows: We first discuss current advancements in related literatures. In Section 3, we explain the learning, network and power models used to develop ns3-fl. Section 4 gives detailed implementation of ns3-fl. In Section 5, we validate our tool with an evaluation of ns3-fl against a real deployment of Raspberry Pis, perform a simulation to emulate 40 Raspberry Pis placed around UCSD under different data, algorithm, and network configurations, and we analyze the performance of our tool, in terms of real clock execution time for various FL rounds.

2 RELATED WORK

2.1 Federated Learning

Federated learning is a type of machine learning which keeps data local to clients, rather than transmitting data to the server. Only updates to the learned model are communicated throughout the network. The baseline for much of the recent research in the field is FedAvg [16] which is an algorithm for synchronous FL. Recent work includes developing client selection policies [22] and manipulating resource allocation to improve convergence [2, 25].

Asynchronous FL is another form of FL which exploits parallelism in the network. The benefit of asynchronous FL over synchronous FL is that the training is not delayed due to stragglers in the network [24]. FedAsync is a novel FL algorithm with similar convergence performance to FedAvg [24]. Many recent works on asynchronous FL focus on node selection algorithms [5, 12], weighted aggregation [3, 4], and semi-asynchronous FL [8] to address device heterogeneity and prevent stale models from decreasing the accuracy of the global model.

2.2 Federated Learning Simulators

The existing FL simulators provide support for algorithmic development, without considering complex network behaviors. Google's FedJAX aims to provide ease-of-use for algorithmic development and provides standard datasets, models, and algorithms for users [21]. FLPrivacy has configurable privacy scenarios and static network settings, such as which clients will drop out and latencies for each client [17].

Flsim is a simulator that was developed to support experimental research, and was created for and first used in [22]. While flsim does not contain any network elements, we link flsim with the ns-3 network simulator to consider the effects of real networks on federated learning, which has not been realized by existing simulators.

2.3 ns-3 Network Simulator

ns-3 is a discrete event network simulator, created for research and educational use [10]. It is an open source project that is licensed under GNU GPLv2. ns-3 supports both ethernet and wireless communication models and can be integrated with testbeds.

Two closely related ns-3 applications to our application are ns3-gym and ns3-ai. ns3-gym supports the integration of OpenAI gym, a toolkit for the development of reinforcement learning algorithms, into ns-3, by using a socket for communication between these processes [7]. ns3-ai also supports the integration of Python-based reinforcement learning frameworks into ns-3, by using a shared memory space for communication between processes [26]. Both ns3-gym and ns3-ai emulate reinforcement learning-based applications over wireless networks, while the proposed ns3-fl is designed for FL applications.

3 MODEL

In this section, we introduce the learning, network and power models used to develop ns3-fl. Note, that the users can easily extend these models in the future to match their research needs.

3.1 Learning Model

The canonical setting in FL is a star-topology network. Suppose there are N client nodes connected to the server. The clients each have a local dataset, which they train on and do not transmit over the network. The goal of FL is to train the global model w with the local data from each client. We define loss function $f(w; x^i, y^i)$ as an error function of how well the model w performs with respect to sample (x^i, y^i). The objective of FL is

$$\min_w F(w) = \frac{1}{N}\sum_{i=1}^{N} F_i(w) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{n_i}\sum_{k=1}^{n_i} f(w; x_k^i, y_k^i) \quad (1)$$

where $F_i(\cdot)$ is the local objective that measures the empirical loss at client i holding n_i local samples.

We support two types of FL algorithms: synchronous and asynchronous. We implement the FedAvg and FedAsync algorithms [16, 24] with their procedures shown in Algorithm 1 and 2 respectively. In each global round $t = \{1, ..., T\}$, both algorithms select a subset of clients $S_t \subseteq \{1, ..., N\}$ to perform local training. We use $|S_t|$ to denote the number of selected clients.

In a synchronous FL training round, the server first transmits the global model to selected clients. When the selected clients receive the model, they update the model and send the updated model to the server. After receiving an updated model from each selected client, the server aggregates the updated client models and updates the global model. This concludes a synchronous FL round.

In an asynchronous FL training round, the server selects a subset of the clients to transmit the global model to. The clients then update global model and transmit their updates to the server. The server updates the global model with the client updates as they are received. The server then transmits the updated global model back

Algorithm 1 FedAvg

```
1: for t = 0, 1, …, T do
2:     Server selects a subset $S_t$ of clients.
3:     Server sends $w^t$ to the clients in $S_t$.
4:     Each client $i$ in $S_t$ updates the $w^t$ for $E$ epochs of stochastic gradient descent (SGD) on $F_i$, creating $w_i^{t+1}$.
5:     Each client $i$ in $S_t$ sends $w_i^{t+1}$ back to the server.
6:     The server aggregates updates from all clients in $S_t$, where $w^{t+1} = 1/|S_t| \sum_{i=1}^{|S_t|} w_i^{t+1}$
```

Algorithm 2 FedAsync

```
1: for t = 0, 1, …, T do
2:     Server selects a subset $S_t$ of clients.
3:     Server sends $w^t$ to the clients in $S_t$.
4:     Each client $i$ in $S_t$ updates the $w^t$ for $E$ epochs of SGD on $F_i$, creating $w_i^{t+1}$.
5:     Each client $i$ in $S_t$ sends $w_i^{t+1}$ back to the server.
6:     Server receives $w_i^{t+1}$ from client $i$, and updates global model, where $w^{t'} = (1-\alpha)w^t + \alpha w_i^{t+1}$, and $\alpha$ is a staleness parameter.
7:     Server sends $w^{t'}$ to client $i$, and client $i$ repeats process from line 4.
8:     When server receives $w_i^{t+1}$ from all clients in $S_t$ at least once, $w^{t+1} = w^{t'}$
```

to the client that communicated their update to the server. This enables faster clients to update the global model multiple times per round while waiting for stragglers. Once each of the selected clients have sent an updated model to the server at least once, the round will end.

3.2 Network Model

ns3-fl relies on ns-3 for network simulations. The pertinent network data that is sent from ns-3 to flsim is the client id, latency, and throughput for each client participating in the current round. Both the ns-3 and flsim processes contain identical networks (i.e. the same number of clients); thus the client ids are the same across both networks. To compute the latency of each round for a client i, we take the latency L_i (seconds) to be

$$L_i = t_u + t_d + t_c. \tag{2}$$

t_u is the uplink time: the time it takes for the client to send the entire updated model to the server. t_d is the downlink time: the time it takes for the client to establish a connection with the server and receive the full global model from the server. t_c is the computational delay: the time it takes for a machine learning algorithm to update the local model.

ns3-fl evaluates the throughput T_i (kbps) for a client i as one of the metrics, which is defined as:

$$T_i = b_r / t_u. \tag{3}$$

Here b_r represents the size of the model to be exchanged between the server and clients.

3.3 Power Model

We define a power model to calculate the energy consumption over the FL training process. Different from the well-known CPU power models based on system variables [1, 27], our model considers machine-learning specific parameters, e.g., the number of multiply–accumulate (MAC) operations. We model the energy used by the client to transmit the model to the server, and the client's energy consumed during computation. Our model is defined for two devices, Raspberry Pi (RPi) 4 [20] and 400 [19], which represent the latest edge computing platforms.

We define the transmission energy E_{tx_i} as:

$$E_{tx_i} = t_{tx_i} \cdot P_{tx_i}, \tag{4}$$

where t_{tx_i} is the time taken to transmit the model from the client to the server obtained from ns-3. The value of P_{tx_i} is based on the average power during transmission for RPi 4s and 400s.

We define the computational energy E_{c_i} as:

$$E_{c_i} = t_{c_i} \cdot P_{c_i}, \tag{5}$$

where t_{c_i} is the computational time per round for client i and P_{c_i} is the average power consumed during t_{c_i} on client i.

To calculate the computational energy, we first calculate the computational time and the computational power. We calculate the computational time t_{c_i} for a client i with the following equation:

$$t_{c_i} = a \cdot M_i \cdot e_i / f_i + b, \tag{6}$$

where M_i is the number of MAC operations that it takes for client i to update their local model over 1 epoch, e_i is the number of local epochs, and f_i is the CPU frequency of client i.

Similarly, the computational power P_{c_i} is calculated with the following equation:

$$P_{c_i} = c \cdot M_i \cdot e_i / f_i + d. \tag{7}$$

All constants a, b, c, and d in Equation (6) and (7) are derived by performing a linear regression on the power measurements on RPi 4s and 400s, respectively, when training over various datasets and machine-learning models at various CPU frequency.

4 IMPLEMENTATION

In this section, we first give an overview of ns3-fl. We then describe our implementation of the Client/Server ns-3 applications, and the interprocess communication protocols we use to communicate between ns-3 and flsim.

4.1 Overview

Figure 1 depicts the architecture of ns3-fl at a high level. ns3-fl consists of two main software blocks, flsim and ns-3. Flsim simulates the data distribution and FL while ns-3 simulates the network.

At the beginning of each FL training round, flsim sends a network simulation request to ns-3, and lists the selected clients for the round. ns-3 will then perform the network simulation for that round, and send back the latency and throughput for each client in the round. Flsim uses the network statistics for each client when computing the convergence time and average throughput for the training round.

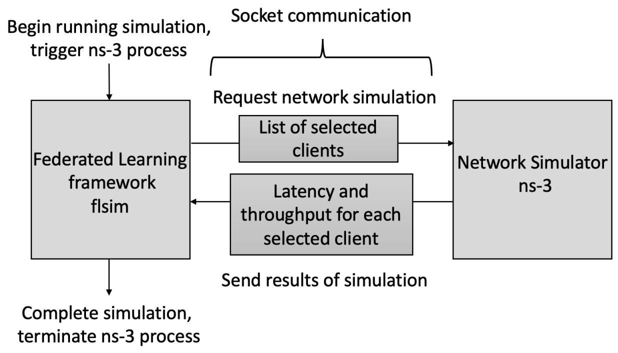
Begin running simulation, trigger ns-3 process
Socket communication
Request network simulation
List of selected clients
Federated Learning framework flsim
Network Simulator ns-3
Latency and throughput for each selected client
Send results of simulation
Complete simulation, terminate ns-3 process

Figure 1: Architectural Overview of ns3-fl

Table 1: Summary of Socket Commands

#	Command	Use
0	RESPONSE	Send results from ns-3 sim
1	STARTSIM	Schedule a round simulation in ns-3
2	EXIT	Terminate ns-3 process
3	ENDSIM	For async, alerts that ns-3 round ended

4.2 Network Implementation

To simulate the network, we create a ClientApplication and a ServerApplication.

The ClientApplication has one mode of operation. At the start of a simulation, the ClientApplication will attempt to connect to the server. Once connected, the ClientApplication will expect to receive the global model from the server. When the ClientApplication receives the full model from the server, we schedule a computational delay. Once the computational delay concludes, the client begins sending the full model back to the server.

The ServerApplication has two modes of operation, synchronous and asynchronous, depending on the type of FL algorithm being simulated.

For synchronous FL, when a client connects to the ServerApplication, the ServerApplication will send the global model to the client. The ServerApplication then waits for the client to send back the updated model. Upon receiving the updated model, the ServerApplication will close the communication socket. Once all the sockets are closed, the latency and throughput for each client in the round will be sent to flsim.

For asynchronous FL, differently, once the ServerApplication receives the full model from the client, the latency and throughput for the client will be *immediately* sent to flsim. If the client was the slowest client in the round, the simulation will end. Otherwise, the server will start a new asynchronous round by sending another model to the client and the process will repeat.

4.3 Communication Protocols

To configure the communication between the ns-3 process and the flsim process, we define four socket commands. Table 1 summarizes these commands.

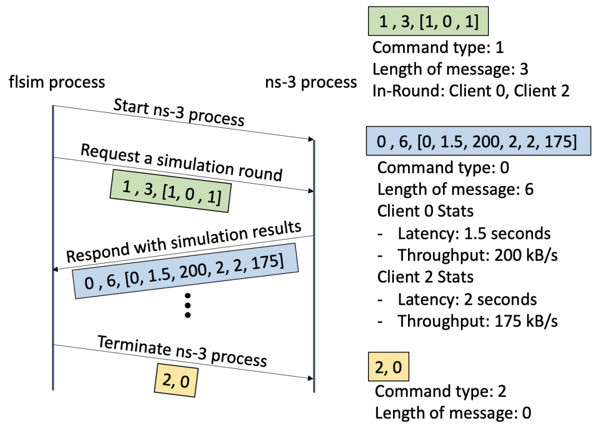

Figure 2: Sync FL Process Communication

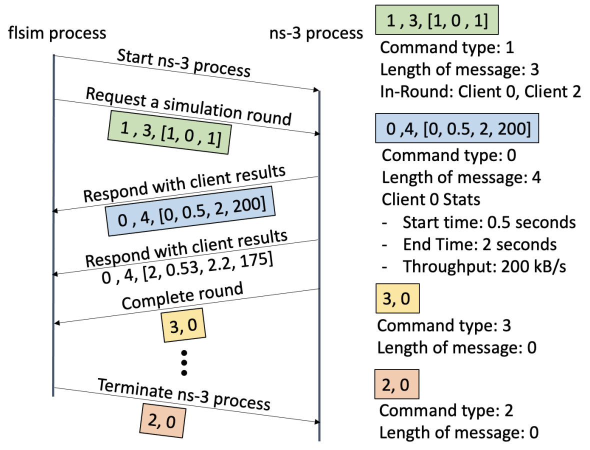

Figure 3: Async FL Process Communication

To schedule a simulation in the ns-3 process, flsim will send a message with a command *STARTSIM* and an array of the form

$$[0, 0, 1, 0, 0]$$

where the length of the array is the number of clients in the simulation, and the 1s represent selected clients for the communication round. The flsim process will then wait for results from ns-3. In the synchronous FL simulation, the ns-3 simulation will send a message with command *RESPONSE*, containing the network statistics for all the selected clients, containing a list of the form

$$[i, L_i, T_i, j, L_j, T_j, \dots]$$

where i and j represent the IDs of the clients in the round. In the asynchronous FL simulation, the ns-3 process will send a message each time the server receives an updated model from the clients, until the slowest client completes the round. The message will be of command type *RESPONSE*, containing the network statistics for a single client:

$$[i, s_i, e_i, T_i]$$

where s_i is the start time of the downlink from server to client, and e_i is the end time of the uplink from client to server. When the slowest client finishes the round, the simulation will send the slowest client's data, and then a message of command *ENDSIM*, to alert the flsim that the round has concluded. When the flsim process finishes training, a message with command *EXIT* will be sent to the ns-3 process to alert that the process should terminate. Figures 2 and 3 depict the concrete usage of the different commands.

5 EVALUATION

In this section, we present the experimental setup, the validation of ns3-fl on a small-scale real deployment based on Raspberry Pis (RPis) and the simulation results on large-scale deployments on the UCSD campus.

5.1 Experimental Setup

We validate ns3-fl on a small-scale real deployments with RPis. We use a RPi400 [19] as the central server, and set up 6 clients consisting of 4 RPi4's [20] and 2 RPi400's. All platforms are equipped with 4GB RAM and connected to the same local Wi-Fi network. Our implementation is based on FedML [9] which is the state-of-the-art FL benchmark. The computational and transmit energy is measured with the Hioki 3334 powermeter [11].

We experiment with ns3-fl under various dataset, learning models and networks settings.

Dataset. In the evaluation, the training task is image classification and we run our simulations with the MNIST [6], FashionMNIST [23] and CIFAR-10 [14] datasets. MNIST is a commonly-used handwriting image classification dataset. FashionMNIST is proposed as a more challenging replacement for MNIST, presenting a cloth classification task. Both MNIST and FashionMNIST have 10 classes and hold 60k samples of 28×28 gray-scale images. CIFAR-10 consists of 60k 32×32 color images in 10 classes, with 6000 images per class. For our experiments, we assign 600 samples to each client following the independently and identically distributed (**IID**) or **non-IID** data distribution. In the IID case, the samples on each client follow the same distribution as the whole dataset, while in the non-IID case we only sample from one class on each client with uniform label distribution. All test data is stored on the server and is used for accuracy evaluation.

Learning Models. We set up a Convolutional Neural Network (CNN) for each dataset. The CNN for MNIST consists of two convolutional layers with 5×5 kernels and two fully-connected layers. The CNN used for FashionMNIST consists of two convolutional layers with 5×5 kernels, each followed by a batch normalization layer, a Rectified Linear Unit (ReLU) activation layer and a max pooling layer. The output is reshaped and fed into a fully-connected layer. Finally, we construct the CNN for CIFAR-10 using two convolutional layers with 5×5 kernels and three fully-connected layers. In the large-scale simulation, we report the accuracy and throughput under various **local epochs** which control the trade-off between convergence time and communication cost.

Networks Settings in ns-3. We set up **a wireless and an ethernet network** in ns-3 for the large-scale simulations with detailed setup summarized in Table 2. To adjust the quality of the network, we vary the client datarate between 80 kbps to 2048 kps. To add loss to the Wi-Fi network, we use the `RandomPropagationLossModel` and `YansErrorRateModel` in ns-3.

Table 2: ns-3 Simulation Parameters

Parameter	Ethernet	Wi-Fi
Routing Protocol	Static Routing	
MAC Layer	CSMA	802.11b
Traffic Type	TCP	
Client Data Rate	80-2048 kbps	
Server Data Rate	100 Mbps	
Packet Size	1024 bytes	
Loss Model	-	`RandomPropagationLossModel`
Error Rate Model	-	`YansErrorRateModel`

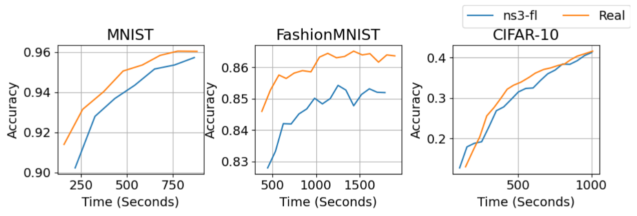

Figure 4: Convergence Validation of ns3-fl against a Real RPi Deployment with 6 Clients and 1 Server on Various Datasets

5.2 Validation with Real Deployment

To demonstrate the validity of ns3-fl, we compare the results of synchronous FL (i.e., FedAvg) with an IID data distribution on a real RPi-based deployment with our tool. We configure the same network and training model in ns3-fl. For both the deployment and the simulation, we train for 20 rounds and 5 local epochs.

Convergence Validation. The accuracy curves are shown in Figure 4. For MNIST, when the training concludes, there is a 0.95% difference between the convergence times and a 0.26% difference between the accuracies of the global model. For FashionMNIST, there is a 0.13% difference between the convergence times and a 1.63% difference between the accuracies of the global model. For CIFAR-10, there is a 0.49% difference between the convergence times and a 0.74% difference between the accuracies of the global model.

The difference in convergence time between ns3-fl and the real deployment can be attributed to variation in the delay of initial client connection to the server in the real deployment. For MNIST and FashionMNIST, the accuracy of the global model is lower on the first round of training for ns3-fl, but the accuracy curves have the same slope and shape as the real deployment for the remaining rounds.

Power Validation. To validate our power model, we compare the computational time, computational energy, and transmit energy calculated by our simulator against a real deployment with 6 clients. Note, that the transmit time is obtained from ns-3 network simulations. Tables 3-5 compare the results from ns3-fl and the real deployment, which shows that ns3-fl is able to approximate the energy consumption in real deployments.

For the computational time, the computed computation time by ns3-fl has an 11.47%, 6.65%, 10.64% percent difference compared to the real deployment when training on a RPi400 over MNIST, FashionMNIST, and CIFAR-10, respectively. For a RPi4, the percent

Table 3: The Mean and Standard Deviation of Computational Time (Seconds) from ns3-fl and Real Measurements on RPi 4 and 400, with CPU Frequency of 1500 MHz, on Various Datasets

Experiment	MNIST	FashionMNIST	CIFAR-10
Real, RPi 400	18.32 ± 2.12	29.69 ± 1.10	33.86 ± 6.57
ns3-fl, RPi 400	20.55 ± 1.1E-5	27.78 ± 1.9E-5	30.44 ± 1.4E-5
Real, RPi 4	18.91 ± 0.84	27.85 ± 1.17	30.30 ± 1.47
ns3-fl, RPi 4	19.01 ± 1.1E-5	27.44 ± 1.9E-5	30.15± 1.4E-5

Table 4: The Mean and Standard Deviation of Computational Energy (J) from ns3-fl and Real Measurements on RPi 4 and 400, with CPU Frequency of 1500 MHz, on Various Datasets

Experiment	MNIST	FashionMNIST	CIFAR-10
Real, RPi 400	81.03 ± 8.92	129.89 ± 5.55	141.00 ± 26.56
ns3-fl, RPi 400	90.05 ± 4.9E-5	125.37 ± 8.5E-5	137.01 ± 6.3E-5
Real, RPi 4	94.10 ± 3.66	136.76 ± 5.02	144.69 ± 6.90
ns3-fl, RPi 4	93.87 ± 5.3E-5	133.25 ± 9.1E-5	143.65 ± 6.7E-5

Table 5: The Mean and Standard Deviation of Transmit Energy (J) from ns3-fl and Real Measurements on RPi 4 and 400, with CPU Frequency of 1500 MHz, on Various Datasets

Experiment	MNIST	FashionMNIST	CIFAR-10
Real, RPi 400	4.68 ± 0.02	0.44 ± 0.02	0.74 ± 0.02
ns3-fl, RPi 400	4.62 ± 1.6E-5	0.45 ± 1.7E-5	0.69 ± 1.9E-5
Real, RPi 4	5.42 ± 0.02	0.49 ± 0.06	0.86 ± 0.03
ns3-fl, RPi 4	5.61 ± 1.7E-5	0.47 ± 1.7E-5	0.81 ± 2.3E-5

difference is 0.53%, 1.48%, 0.50% for MNIST, FashionMNIST, and CIFAR-10, respectively.

The percent difference for the computational energy is 10.54%, 3.54%, 2.87% for a RPi400 when training over MNIST, FashionMNIST, and CIFAR-10. For the RPi4, there is a percent difference of 0.24%, 2.60%, 0.72% when training over MNIST, FashionMNIST, and CIFAR-10, respectively.

For the transmit energy, the percent difference is 1.29%, 2.25%, 6.99% for an RPi400 when training over MNIST, FashionMNIST, and CIFAR-10. For the RPi4, there is a percent difference of 3.45%, 4.12%, 5.99% when training over MNIST, FashionMNIST, and CIFAR-10, respectively.

It can be observed that computational time and energy estimated by ns3-fl has a greater percent difference for the RPi400. When creating the power model, computational time measurements for the RPi400 had an average variance of 6.02 seconds, and the RPi4 had an average variance of 3.10 seconds. Thus, the larger percent difference in the RPi400 computational time and energy estimations can be attributed to the variance in the initial measurements.

5.3 Simulation Results on Large-Scale Deployments

We set up a simulation to emulate training over 40 Raspberry Pis, placed on the UCSD campus. Our topology is shown in Figure 5. We utilize ns3-fl to test performance under different data distributions and network conditions, as well as to refine the learning algorithm used. All of these configurations are common decisions to be made during setting up a federated-learning application. We show that ns3-fl can assist decision making which involves complex trade-offs between data, algorithm and network.

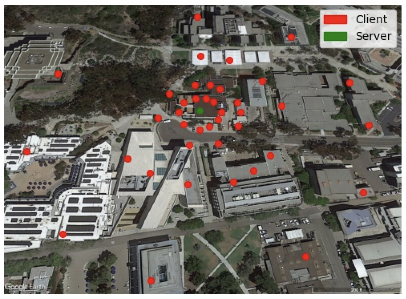

Figure 5: UCSD Campus Topology used in Large-Scale Simulation

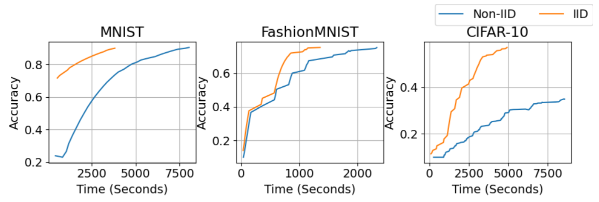

Figure 6: Convergence of ns3-fl when Training on IID v. Non IID Data Distribution over a Wi-Fi Network

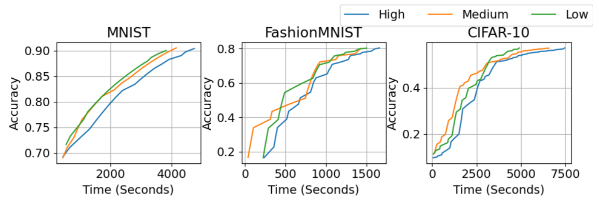

Figure 7: Convergence of ns3-fl when Training on a Low-Loss, Medium-Loss and High-Loss Wi-Fi Network

IID vs. Non-IID Data Distribution. Figure 6 displays the results for training on IID and Non-IID data distribution over a Wi-Fi network. In general, training over IID data leads to a faster convergence time. There is an 83% increase in the convergence time for training when the data is non-IID for MNIST and a 53% increase for FashionMNIST. It takes 12 more rounds for both MNIST and FashionMNIST to converge on non-IID data than IID data. When the data is non-IID for CIFAR-10, the training does not converge. ns3-fl captures the converging behavior of FL under different data distributions.

Network Quality Variations. Figure 7 displays the results of training over a low loss, medium loss, and high loss Wi-Fi network with an IID data distribution. When there is more loss in the network, the convergence time increases when training on MNIST, FashionMNIST, and CIFAR-10. The increase in convergence time is caused by increased transmission time to send the full model for each client. In the high loss network, training on CIFAR-10 has the largest increase in convergence time because converging on

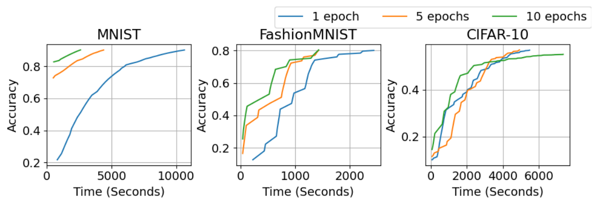

Figure 8: Convergence of ns3-fl when Varying the Local Epochs Per Round over a Wi-Fi Network

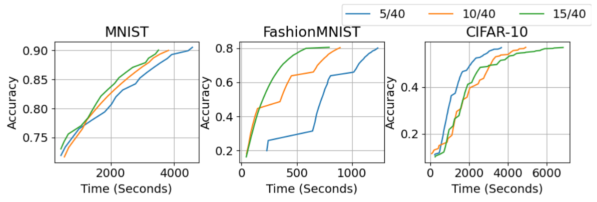

Figure 9: Convergence of ns3-fl when Selecting 5, 10, or 15 Clients Per Round over a Wi-Fi Network

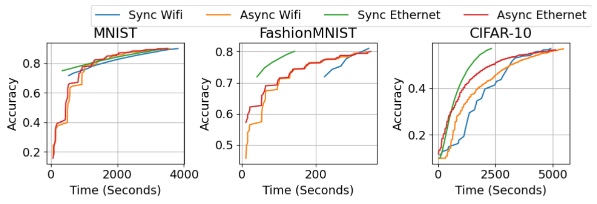

Figure 10: Convergence of ns3-fl when using Synchronous and Asynchronous FL over an Ethernet and Wi-Fi Network

CIFAR-10 requires more training rounds than the other datasets, and thus the transmission delay accumulates. The results prove ns3-fl's ability to capture the effect of network quality on FL performance.

Local Epoch Variation. Figure 8 displays the results of varying the number of local epochs over a Wi-Fi network with an IID data distribution. More local epochs accelerate FL since less global communication rounds is needed, but may risk convergence in challenging datasets. When training over FashionMNIST, the convergence time decreases by roughly 40% when performing 5 or 10 local epochs compared to 1 local epoch. When training over MNIST, the convergence time decreases by 58.46% when performing 5 local epochs compared to 1 local epoch, and by 40.08% when performing 10 local epochs compared to 5 local epochs. When training over CIFAR-10, the convergence time decreases by 10.91% when performing 5 local epochs compared to 1 local epoch. When performing 10 local epochs, the training on CIFAR-10 did not converge. ns3-fl accurately captures the benefits and risks when varying local epochs in FL.

Selected Clients Per Round. Figure 9 displays the results of varying the number of selected clients per round over a Wi-Fi network with an IID data distrubution. For both MNIST and FashionMNIST, when less clients are selected per round, the convergence time and the number of training rounds needed increases. For CIFAR-10,

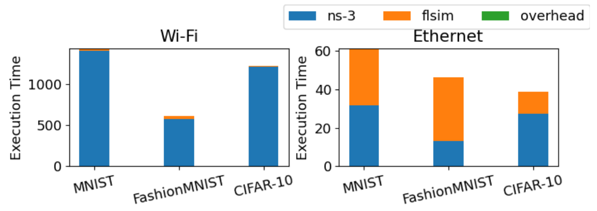

Figure 11: Breakdown of Execution Time (Seconds) of ns3-fl on a Synchronous FL Round over a Wi-Fi Network and an Ethernet Network

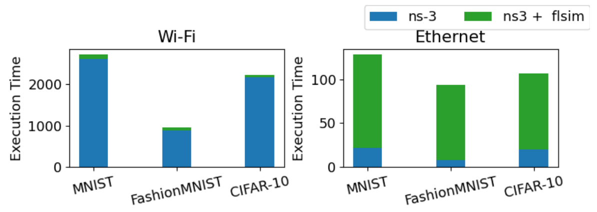

Figure 12: Breakdown of Execution Time (Seconds) of ns3-fl on an Asynchronous FL Round over a Wi-Fi Network and an Ethernet Network

the convergence time increases when more clients are selected. This is because training on CIFAR-10 takes roughly 50 more rounds to converge than the other datasets; thus, the transmission delay for 15 clients outweighs the increase in rounds for 5 clients. The above results serve as a reference when determining the number of clients to select in a FL deployment.

Synchronous vs. Asynchronous FL. Figure 10 displays the results of running the synchronous and asynchronous FL algorithm over an Ethernet and Wi-Fi network (as shown in Table 5) with an IID data distribution. When training on MNIST and FashionMNIST, on the Ethernet network, the synchronous and asynchronous FL algorithms have similar performance. For the Wi-Fi network, the asynchronous FL algorithm outperforms the synchronous FL algorithm. When training on CIFAR-10, synchronous FL over the ethernet network significantly outperforms asynchronous FL, and both algorithms have similar performance over the Wi-Fi network. The above results can aid in determining whether to use synchronous FL or asynchronous FL in a deployment based on the network type.

5.4 Simulation Performance

We also evaluate the execution time breakdown of ns3-fl for overhead analysis. We run the simulations on an Ubuntu Linux system that has an i7-660U dual-core with 8 GB of RAM, and a CPU frequency of 2.6GHz.

Figures 11 and 12 display the breakdown of execution time between ns-3 and flsim under different FL scenarios. The bottleneck in the execution time is the ns-3 simulation for all FL scenarios. Additionally, the ns-3 simulation takes longer for the MNIST and CIFAR-10 datasets because they have significantly larger models than the FashionMNIST dataset. The flsim execution time is similar across both network types for synchronous and asynchronous FL.

The overhead time, which consists of data processing for socket communication and socket communication itself, is minuscule compared to the ns-3 and flsim execution time.

Comparing the breakdowns of the execution time for a FL simulation over a Wi-Fi network and a FL simulation over an Ethernet network, it is evident that the Wi-Fi network simulation takes longer to run for both synchronous and asynchronous FL. This could be attributed to the increased complexity in the Wi-Fi network settings, compared to the Ethernet settings.

For the asynchronous FL simulations, the ns-3 simulation and flsim simulation are able to run simultaneously. This is because of the interprocess communication for asynchronous FL which sends network statistics to flsim as they are generated, instead of waiting until the end of the network simulation as we do in a synchronous FL round. The ns-3 simulations take longer for asynchronous FL because the fastest client in a round must update the global model up to 10 times before we accept a client has dropped out. In the synchronous round, the simulation will end when there are no more scheduled events and we do not dynamically schedule more transmissions, thus we do not need to consider the effect of client drop outs on the simulation.

6 CONCLUSION

In this paper, we introduce ns3-fl, a FL simulator that considers data, algorithm, and network. We build on an existing FL simulator, flsim, and add a network component by connecting flsim with ns-3. Additionally, we create a power model to calculate the energy consumption of FL training on clients in a network. We also demonstrate a use-case of ns3-fl, in aiding decision designs for FL deployments. Our tool was created with FL researchers in mind. We believe that ns3-fl will assist in the practical validation of existing theoretical FL research, strengthening the reliability and tolerance of FL algorithms under real deployment conditions.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable feedback. This work was supported by National Science Foundation under Grants #2003279, #1826967, #2100237, #2112167, and #2112665.

REFERENCES

[1] Aaron Carroll and Gernot Heiser. 2010. An Analysis of Power Consumption in a Smartphone. In *2010 USENIX Annual Technical Conference (USENIX ATC 10)*.

[2] Mingzhe Chen, H. Vincent Poor, Walid Saad, and Shuguang Cui. 2021. Convergence Time Optimization for Federated Learning Over Wireless Networks. *IEEE Transactions on Wireless Communications* 20, 4 (2021), 2457–2471. https://doi.org/10.1109/TWC.2020.3042530

[3] Yujing Chen, Yue Ning, Martin Slawski, and Huzefa Rangwala. 2020. Asynchronous Online Federated Learning for Edge Devices with Non-IID Data. In *2020 IEEE International Conference on Big Data (Big Data)*. 15–24. https://doi.org/10.1109/BigData50022.2020.9378161

[4] Yang Chen, Xiaoyan Sun, and Yaochu Jin. 2020. Communication-Efficient Federated Deep Learning With Layerwise Asynchronous Model Update and Temporally Weighted Aggregation. *IEEE Transactions on Neural Networks and Learning Systems* 31, 10 (2020), 4229–4238. https://doi.org/10.1109/TNNLS.2019.2953131

[5] Zheyi Chen, Weixian Liao, Kun Hua, Chao Lu, and Wei Yu. 2021. Towards Asynchronous Federated Learning for Heterogeneous Edge-Powered Internet of Things. *Digital Communications and Networks* 7, 3 (2021), 317–326. https://doi.org/10.1016/j.dcan.2021.04.001

[6] Li Deng. 2012. The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142. https://doi.org/10.1109/MSP.2012.2211477

[7] Piotr Gawlowicz and Anatolij Zubow. 2018. ns3-gym: Extending OpenAI Gym for Networking Research. (2018). arXiv:1810.03943 [cs.NI]

[8] Jiangshan Hao, Yanchao Zhao, and Jiale Zhang. 2020. Time Efficient Federated Learning with Semi-Asynchronous Communication. In *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*. 156–163. https://doi.org/10.1109/ICPADS51040.2020.00030

[9] Chaoyang He, Songze Li, Jinhyun So, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, Li Shen, Peilin Zhao, Yan Kang, Yang Liu, Ramesh Raskar, Qiang Yang, Murali Annavaram, and Salman Avestimehr. 2020. FedML: A Research Library and Benchmark for Federated Machine Learning. *arXiv preprint arXiv:2007.13518* (2020).

[10] Thomas R Henderson, Mathieu Lacage, George F Riley, Craig Dowell, and Joseph Kopena. 2008. Network Simulations with the ns-3 Simulator. *SIGCOMM demonstration* 14, 14 (2008), 527.

[11] Hioki3334 Powermeter. [n.d.]. Hioki3334 Powermeter. https://www.hioki.com/en/products/detail/?product_key=5812.

[12] Chung-Hsuan Hu, Zheng Chen, and Erik G. Larsson. 2021. Device Scheduling and Update Aggregation Policies for Asynchronous Federated Learning. In *2021 IEEE 22nd International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. 281–285. https://doi.org/10.1109/SPAWC51858.2021.9593194

[13] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2021. Advances and Open Problems in Federated Learning. *Foundations and Trends® in Machine Learning* 14, 1–2 (2021), 1–210.

[14] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning Multiple Layers of Features from Tiny Images. (2009).

[15] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated Learning: Challenges, Methods, and Future Directions. *IEEE Signal Processing Magazine* 37, 3 (2020), 50–60.

[16] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Artificial Intelligence and Statistics*. PMLR, 1273–1282.

[17] Vaikkunth Mugunthan, Anton Peraire-Bueno, and Lalana Kagal. 2020. PrivacyFL: A Simulator for Privacy-Preserving and Secure Federated Learning. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management* (Virtual Event, Ireland) *(CIKM '20)*. Association for Computing Machinery, New York, NY, USA, 3085–3092. https://doi.org/10.1145/3340531.3412771

[18] Dinh C. Nguyen, Ming Ding, Pubudu N. Pathirana, Aruna Seneviratne, Jun Li, and H. Vincent Poor. 2021. Federated Learning for Internet of Things: A Comprehensive Survey. *IEEE Communications Surveys and Tutorials* 23, 3 (2021), 1622–1658. https://doi.org/10.1109/comst.2021.3075439

[19] Raspberry Pi 400. [n.d.]. Raspberry Pi 400. https://www.raspberrypi.com/products/raspberry-pi-400/.

[20] Raspberry Pi 4B. [n.d.]. Raspberry Pi 4B. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.

[21] Jae Hun Ro, Ananda Theertha Suresh, and Ke Wu. 2021. FedJAX: Federated Learning Simulation with JAX. *CoRR* abs/2108.02117 (2021). arXiv:2108.02117 https://arxiv.org/abs/2108.02117

[22] Hao Wang, Zakhary Kaplan, Di Niu, and Baochun Li. 2020. Optimizing Federated Learning on Non-IID Data with Reinforcement Learning. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 1698–1707.

[23] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms. arXiv:1708.07747 [cs.LG]

[24] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. 2019. Asynchronous Federated Optimization. *ArXiv* abs/1903.03934 (2019).

[25] Zhaohui Yang, Mingzhe Chen, Walid Saad, Choong Seon Hong, and Mohammad Shikh-Bahaei. 2021. Energy Efficient Federated Learning Over Wireless Communication Networks. *IEEE Transactions on Wireless Communications* 20, 3 (2021), 1935–1949. https://doi.org/10.1109/TWC.2020.3037554

[26] Hao Yin, Pengyu Liu, Keshu Liu, Liu Cao, Lytianyang Zhang, Yayu Gao, and Xiaojun Hei. 2020. Ns3-Ai: Fostering Artificial Intelligence Algorithms for Networking Research. *Proceedings of the 2020 Workshop on ns-3* (2020), 57–64.

[27] Lide Zhang, Birjodh Tiwana, Zhiyun Qianand Zhaoguang Wang, Robert P Dick, Zhuoqing Morley Mao, and Lei Yang. 2010. Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. 105–114.