SAFA: A Semi-Asynchronous Protocol for Fast Federated Learning With Low Overhead

Wentai Wu, *Student Member, IEEE*, Ligang He, *Member, IEEE*, Weiwei Lin, Rui Mao, Carsten Maple, and Stephen Jarvis, *Member, IEEE*

Abstract—Federated learning (FL) has attracted increasing attention as a promising approach to driving a vast number of end devices with artificial intelligence. However, it is very challenging to guarantee the efficiency of FL considering the unreliable nature of end devices while the cost of device-server communication cannot be neglected. In this article, we propose SAFA, a semi-asynchronous FL protocol, to address the problems in federated learning such as low round efficiency and poor convergence rate in extreme conditions (e.g., clients dropping offline frequently). We introduce novel designs in the steps of model distribution, client selection and global aggregation to mitigate the impacts of stragglers, crashes and model staleness in order to boost efficiency and improve the quality of the global model. We have conducted extensive experiments with typical machine learning tasks. The results demonstrate that the proposed protocol is effective in terms of shortening federated round duration, reducing local resource wastage, and improving the accuracy of the global model at an acceptable communication cost.

Index Terms—Distributed computing, machine learning, edge intelligence, federated learning

1 Introduction

With the prevalence of Internet of Things (IoT), the advance in Machine Learning (ML) techniques stimulates the demand of compute capacity significantly from a broad range of applications which more or less integrate Artificial Intelligent (AI) into the edge and end devices to empower their underlying business logic. By 2022, more than 80 percent of enterprise IoT projects are expected to have AI components embedded [1]. Also, it has been an emerging trend that users are becoming more sensitive to the data privacy protection mechanism of AI applications, while their performance, in many cases, is still expected to be guaranteed in the first place.

It is promising for intelligent applications to learn their models on massively distributed data. However, there are still several obstacles to date. First, it is unrealistic to collect decentralized data constantly from all the end devices and store them in a centralized location, which can probably cause a lot of potential risks (e.g., data leakage) and poses privacy threats to end users. Second, it could be communication-intensive to train a global model using traditional optimization methods, no matter in a cloud-central or a distributed manner. On-cloud centralized training incurs heavy load (as well as big risks) in data transfer when moving the data from the edge of network to the cloud, whilst most distributed optimization approaches incur fairly frequent communications between devices and the cloud in order to exchange gradients (of a mini-batch, typically) and weights. However, in practical circumstances (e.g., edge computing environments), the devices are hardly reliable and the cost of communication can be prohibitive. For example, devices may drop offline intermittently and data transfer is charged in cellular networks.

Privacy concerns may prohibit moving data outside local devices. Machine learning in such environments is challenging due to the following properties: 1) *Unbalanced and biased data distribution*: the end devices may own a variable amount of on-device data and the distribution of the data in different devices may be different; 2) *Massive distribution*: it is usual to see a huge fleet of disparate devices at the edge as participants; 3) *Unreliability*: either the devices themselves and the connection to the cloud are unreliable. End devices could opt out occasionally or go offline unexpectedly. The communication could be expensive.

Federated Learning (FL) [2], [3], a promising framework, was proposed by Google to address the aforementioned challenges. The work presents a distributed solution (i.e., Federated Optimization) to optimizing a global machine learning model without moving data out of local devices, and introduces *FedAvg* as an optimization protocol in federated setting. Rather than collecting gradients from clients (i.e., end devices), FedAvg adopts a different approach, in which multiple iterations of local updates (using gradient descent) are followed by a global aggregation that takes a weighted average of the resulting models from the clients.

- *Wentai Wu, Ligang He, and Stephen Jarvis are with the Department of Computer Science, University of Warwick, CV4 7AL Coventry, United Kingdom. E-mail: {wentai.wu, Ligang.He, S.A.Jarvis}@warwick.ac.uk.*
- *Weiwei Lin is with the School of Computer Science and Technology, South China University of Technology, Guangzhou 510641, China. E-mail: linww@scut.edu.cn.*
- *Rui Mao is with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China. E-mail: mao@szu.edu.cn.*
- *Carsten Maple is with Warwick Manufacturer Group, University of Warwick, CV4 7AL Coventry, United Kingdom. E-mail: CM@warwick.ac.uk.*

Manuscript received 17 Oct. 2019; revised 4 May 2020; accepted 10 May 2020. Date of publication 14 May 2020; date of current version 7 Apr. 2021. (Corresponding author: Ligang He.)
Recommended for acceptance by Ahmed Louri.
Digital Object Identifier no. 10.1109/TC.2020.2994391

An obvious advantage of FedAvg is the reduction of communication frequency. FedAvg and many implementations of FL systems (e.g., [4]) adopt synchronous training protocols to avoid the prohibitive number of updates. Although synchronous protocols seem to be the natural choice for the FL setting, a number of limitations stand out as follows: 1) *Unreliable fraction of effective participants*: in each round, the server selects a fraction of clients randomly to perform local training and expects them to commit their local training results. However, the number of clients which manage to commit their results are very uncertain given the unreliable nature of end devices; 2) *Low round efficiency*: To aggregate the local results at the end of each round, FedAvg has to wait for all selected clients to finish, among which there may be stragglers while the crashed ones may never respond. Consequently, the global learning progress is suspended until a timeout threshold is reached; 3) *Under-utilization of clients*: With random selection, many capable clients are likely to remain idle even if they are ready and willing to participate in the training; 4) *Progress waste*: The selected clients may not finish local training in time, and the progress made could be wasted because the client will be forced to overwrite its local model with the global model next time when the client is selected again.

In this paper, we propose a Semi-Asynchronous Federated Averaging (SAFA) protocol based on FedAvg [3] to achieve fast, lag-tolerant federated optimization. SAFA takes advantage of several efficiency-boosting features from asynchronous machine learning approaches (e.g., [6], [9]) while making use of a refined pace steering mechanism to mitigate the impact of straggling clients and stale models (i.e., staleness [9]) on the global learning progress. Moreover, we adopt a novel aggregation algorithm that exploits a cache structure (in the cloud) to bypass a fraction of client updates so as to improve convergence rate at a low cost of communication. The main contributions of our work are outlined as follows:

- We take into account the unreliability and heterogeneity of end devices and propose a Semi-Asynchronous Federated Averaging (SAFA) protocol to alleviate the staleness, boost efficiency and better utilize the progress made by stragglers.
- We introduce a simple hyper-parameter, *lag tolerance*, to flexibly control the behavior of SAFA protocol. We also empirically analyze the impact of lag tolerance on SAFA by observing how it affects the critical metrics such as synchronization ratio and version variance.
- We conducted extensive experiments to evaluate SAFA on several typical machine learning tasks in multiple FL settings varying from tiny to relatively large-scale edge environments. SAFA is evaluated in terms of several important metrics such as model accuracy, round efficiency and communication cost.

The rest of this paper is organized as follows: Section 2 summarizes some relevant studies on distributed learning and federated learning. In Section 3, we formulate the optimization problem for FL, detail the design of SAFA and analyze the impact of the hyper-parameter in SAFA. In Section 4, we present and discuss the experimental results. We conclude this paper in Section 5.

2 Related Work

The fusion of Edge Computing and Artificial Intelligence (i.e., Edge Intelligence [11], [12]) has emerged as a new focus of research ever since we began to realize the potential benefits of sinking the computation to and outside the edge whilst the increasing capacity of end devices makes it natural to empower them with AI and support the applications such as intelligent surveillance [21] and mobile keyboard prediction [22] at the edge.

Distributed machine learning is believed to be an ideal solution for big data analytics according the rule "moving computation closer to data". However, the majority of distributed ML approaches (e.g., [14], [15], [16]) claim their efficacy based on the conditions such as homogeneity [16], high-performance nodes, ultra-fast connections [17] and so on, which are unrealistic in edge computing or IoT environment. In fact, end devices in an edge environment can be fairly unreliable, highly heterogeneous in performance and have limited communication. The limitation of data access is another prominent issue. Many distributed ML approaches cannot achieve the desired accuracy without making the entire dataset available to every worker. However, it is impossible in many situations to gather the data from a massive number of distributed devices given the expensive communication (via cellular networks) and, most importantly, the data privacy requirements by end users [18].

Federated Learning (FL) [3], first proposed by Google, is a new approach to fitting machine learning into the edge. The survey by Zhou *et al.* [13] summarizes recent studies on edge intelligence and lists FL as one of the most uprising technologies for distributed training at the edge. As the primitive FL protocol, *FedAvg* [3] was designed to perform synchronous optimization in federated settings. Xie *et al.* [9] proposed *FedAsync*, an asynchronous federated optimization scheme that regularizes local optimization and adopts the non-blocking update of the global model. A similar protocol has been exploited by Sprague *et al.* [10] in a geo-spatial application for training a global model asynchronously, allowing the devices to join halfway. However, the main issue of the asynchronous approaches is that the server may receive too many local updates sent from a massive number of clients that remain active, which could overwhelm the server but with little benefit to the model convergence.

In terms of model accuracy, Chen *et al.* [5] experimentally demonstrated that synchronous Stochastic Gradient Descent (SGD) can outperform asynchronous approaches in the data center setting, which to some extent inspired the synchronous design of FL. A number of variants have been proposed to mitigate the deficiencies of FL from different aspects such as round efficiency [20] and communication cost [2]. Wang *et al.* [19] proposed a control algorithm that adaptively determines the interval of global aggregation under a given resource budget. To address the inefficiency of FL under poor wireless channel conditions, Nishio and Yonetani [20] implemented a mobile edge computing (MEC) framework in which a protocol is designed to filter out slow clients based on the estimation of the clients' work time at the selection stage and consequently shorten round length. However, their scheme relies on accurate estimation and does not take the client unreliability into account.

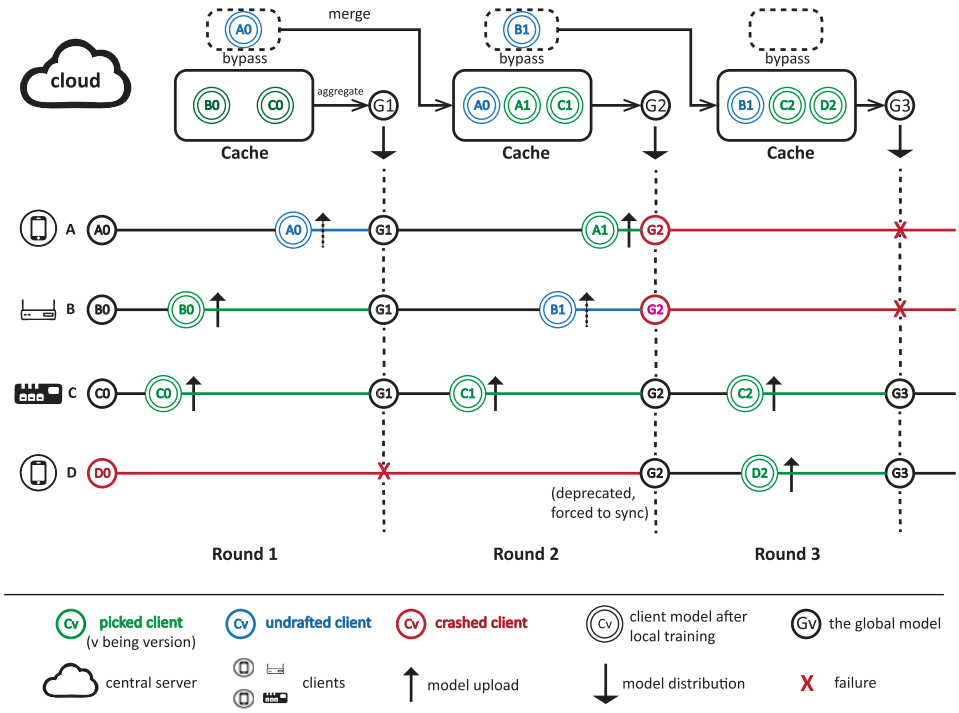


Fig. 1. The diagram of SAFA protocol showing the interaction between the cloud and end clients in different states.

How to speedup the convergence rate of FL remains an open challenge. On this point, we argue that the optimization mechanisms for traditional distributed SGD have great potential in FL. For example, gradient staleness control has been shown critical to guarantee convergence [23]. Dutta *et al.* [24] theoretically characterized the trade-off between reducing error (by including more stragglers) and shortening run time (by bounding staleness). Wang *et al.* [25] refined ASGD by modulating the learning rate based on the staleness of incoming gradients. Smith *et al.* [8] proposed MOCHA, a fault- and straggler-tolerant multi-task learning method without forging a global model. Chen *et al.* [7] introduced backup workers to reduce server waiting time in synchronous stochastic optimization. Inspired by these approaches, we investigate the impact of straggling clients and model staleness on FL, and design a fast FL protocol that is well adapted to unreliable environments.

3 The SAFA Protocol

The proposed Semi-Asynchronous Federated Averaging (SAFA) protocol is designed to solve the global optimization problem as below:

$$\arg\min_{w\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n} f(w; x_i, y_i), \tag{1}$$

where w denotes the parameters of the global model (the number of parameters = d), $f(w; x_i, y_i)$ represents the loss of the inference on sample (x_i, y_i) made by the model with w as its parameters. Note that data samples are distributed among disparate end devices, which are called clients in FL settings. Let M denote the set of m clients, and D_j the partition of data residing in client j, then the target function can be rewritten as

$$\arg\min_{w\in\mathbb{R}^d} \frac{1}{n}\sum_{j=1}^{m}\sum_{i\in D_j} f(w; x_i, y_i). \tag{2}$$

Note that the problem definition here is in accordance with [3], but is different from [9]. Xie *et al.* [9] define their target function as the average of the average loss (on local partitions), which is fair at the local partition level but not the case at the sample level, because data samples in small local partitions take larger weights in their target function.

In this section, we present the workflow of SAFA with the underlying design principles in detail. As a refined FL protocol, SAFA consists of three operations: lag-tolerant model distribution, post-training client selection and discriminative aggregation. A typical FL process driven by SAFA is shown in Fig. 1. We will use this diagram as an example throughout this paper to illustrate our FL training process.

For clarity, we list in Table 1 the symbols frequently used in this paper.

3.1 Lag-Tolerant Model Distribution

In the federated setting, an important guarantee for the convergence of the global model is the quality of local models. The quality of a local model depends on whether local training is sufficient (i.e., training sufficiency) and on the starting model on which local training is based. Training sufficiency can be achieved by allowing adequate local iterations (i.e., epochs), while the version control is a non-trivial task. The original FL algorithm [3] prevents outdated clients with stale models from committing, which simplifies FL process but also throttles the potential of accelerating convergence.

Motivated by the problem, we first present a lag-tolerant model distribution algorithm which does not always enforce synchronization (i.e., allows some clients to stay asynchronous with the cloud) and is tolerant to outdated local models (i.e., staleness). The key idea is to develop a better way to get

TABLE 1
List of Symbols

Symbol	Description
D	the complete dataset
n	the size of D (i.e., $n = \|D\|$)
D_i	the data partition on client i
n_i	the size of client i's local partition
M	the set of clients (i.e., end devices)
m	total number of clients
v_i	the version of client i's local model
M_v	the set of clients whose model version is v
P	the set of picked clients
P_v	the set of picked clients of version v
K	the set of crashed clients
K_v	the set of crashed clients of version v
W	the set of clients that complete local training
Q	the set of undrafted clients
Q_v	the set of undrafted clients of version v
w	parameters of the global model
w_k	parameters of the local model on client k

the stragglers (i.e., clients with stale models) involved in the model aggregation and leverage their progress for faster federated learning. In this paper, we refer *stragglers* to the clients who are slow and still conducting local training based on an outdated model. Normally, the clients are supposed to start epochs of training based on the latest global model received from the server. However, device crashes or network problems generate the stragglers inevitably.

With a version-based criterion, SAFA only requires specific clients to retrieve the latest global model from the server. Before a round of local training starts, the server classifies all clients into three states (or categories) based on their current versions: *Up-to-date*, *deprecated* and *tolerable*, which are defined as follows.

Definition 1 (Up-to-date clients). *the clients that have completed the previous round of local training (and submitted models successfully) are reckoned up-to-date at the start of this round.*

Definition 2 (Deprecated clients). *the clients that still base local training on the models that are too stale compared to the version of the global model.*

Definition 3 (Tolerable clients). *the clients that do not base local training on the latest global model, but the model version they are based on is not too old either. This is a state that stands between* Up-to-date *and* Deprecated.

SAFA only requires the up-to-date and deprecated clients to synchronize with the server, while the tolerable clients stay asynchronous with the server. This is why SAFA is called a semi-asynchronous distributed training scheme. We let up-to-date clients synchronize with the server in order to prevent model divergence [3]. Deprecated clients are forced to synchronize so that the global model will not be poisoned by the seriously outdated local models.

After a round of local training is completed on device, the clients will then be labeled *picked*, *undrafted* or *crashed* based on the result of client selection in SAFA, which is a post-training process. The server tags clients with these labels after the selection quota is met or the round time limit is reached. The picked clients are those whose local training results in this round are selected to be used in the following aggregation step. The undrafted clients are those whose local training results are not selected but still get cached by the server for future use. Crashed clients are those who fail to finish a round of local training - clients can either opt out or drop offline intermittently (i.e., any time during training) with a certain probability (which we refer to as crash probability).

In Fig. 1 we illustrate the workflow of SAFA with four end devices: clients A to D, with which the system is to perform several federated rounds of training. Clients start local training from their local model versioned A0 to D0 (the initial model version for each round of local training is depicted in the figure by a single circle with the model version in the middle). After a client completes its local training, local parameters are updated (depicted by double circles with the model version in it) and are uploaded to the server (depicted by upwards arrows). The server selects submitted results only from a portion of clients (the client portion is 50 percent in this example) to update the global model. The clients whose updates are selected are tagged as picked clients (colored green), for example, clients B and C in the 1st round. The selected updates are placed in a cache structure by the server. The cache maintains the entries of the latest local models uploaded from the picked clients and will be used for aggregation. The clients whose results are not selected are undrafted clients (colored blue), e.g., client A in round 1 and client B in round 2. Updates from these clients are stored in the bypass structure to avoid futile work locally. The clients who cannot complete their local training due to any reason (such as opt-out or network failure) are crashed clients (highlighted red), such as client D in the 1st round.

Each round ends with a new version of global model (i.e., G1 to G3 in this diagram), which, at the start of the next round, will be distributed to (i.e., synchronized with) the up-to-date and deprecated clients. In the first round, for example, A, B and C successfully complete local training and upload their updates (in spite of A being undrafted), thus they become up-to-date clients (i.e., tagged *up-to-date* by the server). The results of undrafted clients will not be merged into the global model in the upcoming aggregation step, but may take effect in future rounds via a bypass structure (squares with dashed lines) that saves these updates temporarily. The bypass will merge with the cache right after the current aggregation step before the next round starts.

In this example, we assume the maximum tolerable version lag is 2. In Fig. 1, client D does not manage to finish local training in two rounds. Therefore, it is tagged *deprecated* and forced to synchronize with the server, which means client D needs to replace its stale local model with the latest global model. To decide whether a local update should be accepted, here we adopt a simple criterion based on the difference between the versions of the global model and the local model, which is called *lag tolerance*. Therefore, the deprecated clients are those whose local version lags behind the version of the global model by more than the specified *lag tolerance*. Specifically, our lag-tolerant distribution principle can be formulated as follows:

$$w_k(t) = \begin{cases} w(t-1) & \text{if } k \in \bigcup_{v=t-1} M_v, \text{or } k \in \bigcup_{v<t-\tau} M_v, \\ & \text{// up-to-date or deprecated clients} \\ w_k(t-1) & \text{if } k \in \bigcup_{t-\tau \le v < t-1} M_v \\ & \text{// tolerable clients} \end{cases}, \quad (3)$$

where $w(t-1)$ denotes the latest global model parameters (i.e., the aggregation result from last round) upon the start of round t, and w_k denotes the parameters of client k's local model; τ stands for *lag tolerance*, which is the only hyper-parameter in SAFA. The lag-tolerant model distribution forces the up-to-date and deprecated clients to adopt the latest global model as the base model for the next round of training, while the tolerable clients can continue to work on their previous local results. The hyper-parameter *lag tolerance* in some ways controls the tradeoff between communication overhead and the convergence rate of federated optimization. If it is set too small, the server may suffer heavy downlink transmission as the portion of deprecated clients increases. If it is set too large, the convergence of the global model could be unsteady. The impact of *Lag tolerance* will be analyzed later with empirical studies.

3.2 Client Selection

An important property of end devices is unreliability, which means that they occasionally drop offline for some reasons such as power outage (or low battery level), inaccessible network or manual shutdown/opt-out of training. In this paper, we refer to these temporarily unavailable states as *crashed*. Every client has a certain probability to crash in each round of training. For clients that stay active and connected to the central server (throughout a round of training), we assume they are always able to finish the task assigned within a certain period of time (otherwise they are also reckoned crashed).

The population of committed updates should be carefully limited considering a huge fleet of end devices [27]. McMahan *et al.* [3] use a hyper-parameter C to control the maximum fraction of clients allowed to participate in one round of training. Moreover, C serves as the criterion in the *FedAvg* [3] protocol by which the server keeps waiting for selected clients to end an global round. In our approach, we retain this hyper-parameter but no longer apply it as a hard constraint. Instead, we release the restriction to allow all clients to participate if they are willing to, and enable the central server to end a round once C-fraction of updates have been received.

Apparently, the efficiency of federated optimization is closely associated to the fraction of picked clients. One may think that we can set C to a large value (e.g., close to 1.0) and pick as many clients into each round as possible. However, it is neither realistic nor beneficial to do so. On the one hand, allowing more clients to participate increases the potential risk of uplink congestion and the communication cost as well. In each round, the server may have to wait for more clients among which some may never respond (because picked clients could crash midway). On the other hand, involving a large number of updates leads to limited benefit to the global model especially in the last few rounds before convergence [3].

It is notable that the fraction of selected clients (called selection fraction) is not equivalent to the actual fraction of clients that finish local training and commit their models in time. In an unreliable environment, picked clients can crash halfway in their training progress or fail to upload their trained models. In this paper, we define a metric termed Effective Update Ratio (EUR) to measure the fraction of effective updates from the local (i.e., all clients) to the cloud (i.e., central server(s))

$$EUR = \frac{|P - P \cap K|}{|M|}, \tag{4}$$

where P and K are the sets of picked and crashed clients, respectively. Obviously EUR is positively correlated with the size of P and negatively correlated with that of K. As mentioned, simply increasing the pick fraction can bring about problems in the FL context, while the crash of clients is not predictable or controllable (improving client stability is beyond the scope of this paper). As a solution, we propose to let the central server collect local update after local training instead of randomly selecting clients at the very beginning of a global round. This means the server does not need to wait for those designated clients for aggregation but are able to execute the aggregation step once it has received a C-fraction of update. Our post-training selection effectively decouples the server with the selected clients and consequently improves EUR, which facilitates faster convergence of the federated optimization. Another advantage of doing so is a significant boost of round efficiency in the case the clients crash with a fairly high probability. Based on the outcome of selection (before the aggregation step is carried out), the server tags the clients with three different labels: *crashed*, *picked* and *undrafted*. Only picked clients are eligible to update its corresponding cache entry right before the aggregation conducted by the central server. Undrafted clients also commit their updates but their updates will bypass the following aggregation.

Considering the "selection-ahead-of-training" scheme used in the synchronous FL (e.g., FedAvg [3]), its effective update ratio, according to Eq. (4), is $C(1 - \frac{|K|}{|M|})$. By contrast, SAFA adopts a "selection-after-training" scheme that theoretically yields the value of EUR as follows:

$$EUR = \begin{cases} 1 - R & \text{if } C \geq 1 - R, \\ C & \text{if } C < 1 - R, \end{cases} \tag{5}$$

where C is the selection fraction and R denotes the crash ratio over all the clients (i.e., $R = \frac{|K|}{|M|}$). Fig. 2 demonstrates how SAFA promotes the effective update ratio in FL - involving as many clients as possible to fulfill the fraction C.

From Fig. 2 and considering (4), we can see a clear improvement of EUR by SAFA, which minimizes the negative impact of clients' failure. Nevertheless, extremely high crash ratio of clients will still cause a low value of EUR even with our selection method. The phenomenon will be analyzed later in our experiment section.

As figured out by Bonawitz *et al.* [4], bias is introduced if each device is equally likely to participate each round because they differ in performance and network access privilege. The problem remains if we merely use the client selection method mentioned above. Therefore, we further propose to alleviate the bias using a compensatory client selection algorithm. The principle is simple – higher priority is given to those clients that are less involved. In each round the server maintains a list of IDs of clients that missed the previous round of training, and their updates will be picked

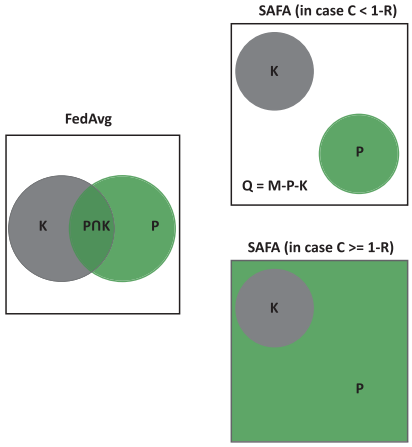


Fig. 2. The diagram shows the client selection policies by FedAvg and SAFA. The selection fraction is C in both policies. SAFA picks C-fraction of clients if no less than C fraction of clients uploaded their local models (i.e., updates). Otherwise it will pick all clients that have committed the local models. A square in the diagram represents the full client set M. K and P are the sets of crashed and picked clients, respectively.

prior to others for the coming aggregation. The pseudo-code of our selection policy is shown in Algorithm 1.

Algorithm 1. Compensatory First-Come-First-Merge (CFCFM) Client Selection

Input: round number t, client set M, last-round picked clients $P(t-1)$, selecting fraction C, round deadline T_{lim}
Output: clients to pick $P(t)$
$P(t) = \varnothing$;
$Q(t) = \varnothing$;
$quota = C \cdot |M|$;
while $|P(t)| < quota$ and $T_{round} < T_{lim}$ **do**
  Await new updates;
  $w'_k \leftarrow$ update arrives from client k;
  if k not in $P(t-1)$ **then**
    add k to $P(t)$;
  else
    add k to $Q(t)$;
  end
end
if $|P(t)| < quota$ **then**
  Sort $Q(t)$ by arrival time;
  $q \leftarrow quota - |P(t)|$;
  $P'(t) \leftarrow$ first q clients in $Q(t)$;
  $Q(t) \leftarrow Q(t) - P'(t)$;
  $P(t) \leftarrow P(t) + P'(t)$;
end
return $P(t)$

In the selection, we stop involving more clients once the quota has been met, namely C-fraction of clients have been selected from $P(t-1) \cap W(t)$. Otherwise the algorithm continues to wait and accept the updates (until a deadline is reached) from the rest of clients which, in practice, will arrive at the cloud successively.

3.3 Discriminative Aggregation

After a round of local training completes, the server has received a collection of updates from the end devices. We adopt three steps to aggregate local updates. The first step is the pre-aggregation cache update, which overwrites the corresponding entries (for storing model parameters) of the selected clients in the cache. In the second step, the updates stored in the cache are aggregated. In the third step, the undrafted updates are placed in the cache, which can be used in the next round of global model aggregation. Since the picked and undrafted updates are treated in a different manner in the aggregation, it is called the three-step discriminative aggregation, which is formally formulated as follows:

(1) *Pre-aggregation Cache Update*:

$$w_k^*(t) = \begin{cases} w'_k(t) & \text{if } k \in P(t), \\ w(t-1) & \text{if } k \in \bigcup_{v<t-\tau} M_v(t), \\ w_k^*(t-1) & \text{otherwise} \end{cases} \tag{6}$$

where $w_k^*(t)$ denotes the kth entry of the cache structure (see Fig. 1), and $w'_k(t)$ denotes the trained local model at round t. Entries of deprecated clients will be replaced with the global model $w(t-1)$.

(2) *SAFA Aggregation*:

$$w(t) = \sum_{k=1}^{m} \frac{n_k}{n} w_k^*(t). \tag{7}$$

(3) *Post-aggregation Cache Update*:

$$w_k^*(t+1) = \begin{cases} w'_k(t) & \text{if } k \in Q(t), \\ w_k^*(t) & \text{otherwise} \end{cases}, \tag{8}$$

where $P(t)$, $Q(t)$, and $K(t)$ denote the sets of picked, undrafted, and crashed clients, respectively in round t.

Algorithm 2. Semi-Asynchronous Federated Averaging (SAFA) Protocol

Input: maximum number of rounds r, client set M, local mini-batch size B, number of local epochs E, learning rate η, lag tolerance τ
Output: finalized global model
Server process: // running on the central server;
Initializes client connections;
Initializes global model $w(0)$ and the cache;
for round $t = 1$ to r **do**
  Distributes $w(t-1)$ according to Eq. (3) given τ;
  for each client k in M **in parallel do**
    $w'_k(t) =$ client_update$(k, w_k(t))$
  end
  Collects and selects client updates using CFCFM;
  Updates cache according to Eq. (6);
  Performs aggregation and get $w(t)$ using Eq. (7);
  Updates cache according to Eq. (8);
end
return $w(r)$
Client process: // running on the client k;
client_update(k, w_k)**:;**
$B_k \leftarrow$ batches from D_k of size B;
for epoch $e = 1$ to E **do**
  for batch b in B_k **do**
    $w_k = w_k - \eta \nabla f(w_k; b)$;
  end
end
$w'_k = w_k$;
return w'_k to the server

For SAFA, there are three cases of changes in the cache after a global around t. For picked clients, their updates will be kept in the cache after being merged into the global model. For undrafted clients, the updates will not take effect in this round but will be carried to the next round by the post-aggregation step. For the crashed clients, their entries stay unchanged only if they have not been deprecated. Otherwise these entries will be replaced by the global model (i.e., $w(t-1)$ in Eq. (6)) to avoid heavy staleness.

Now we can present the complete workflow of the proposed SAFA protocol outlined in Algorithm 2. The server orchestrates the process holistically in rounds. At the beginning of each round, the server first checks the version of clients and distributes the latest global model in a lag-tolerant manner (see Eq. (3)) given the hyper-parameter τ. Then the server begins to listen and collects the updates (i.e., trained local model) from clients. Clients train their native models on local datasets using the gradient descent method. Based on Algorithm 1, the clients missing the previous round will have the priority to be selected to meet the pre-set fraction C. Following the client selection, the server then executes the three-step discriminative aggregation, which merges all the entries in the cache into the global model, i.e., $w(t)$, and updates the cache entries of undrafted clients.

3.4 Analysis of Lag Tolerance

We analyze the impact of *lag tolerance* from different perspectives. As mentioned, this hyper-parameter is crucial to the pace-steering of the SAFA protocol. When *lag tolerance* is small, clients/models become deprecated frequently, resulting in relatively high cost in model distribution. If it is set to a big value, the server will be very tolerant to the stragglers, which will probably cause high variance in the versions of local models and consequently slow down the convergence of the global model. Thus, we introduce two holistic metrics: Synchronization Ratio (SR) and Version Variance (VV). SR measures the usage of downlink by which the global model is distributed to the edge of network. VV is defined based on the version distribution of local updates. For SAFA, we formulate SR and VV as follows:

$$SR_{SAFA} = \frac{1}{rm}\sum_{t=1}^{r}(|\bigcup\nolimits_{v=t-1} M_v(t)| + |\bigcup\nolimits_{v<t-\tau} M_v(t)|), \tag{9}$$

where r is the number of global rounds and m is the number of clients. SR is calculated based on our lag-tolerant distribution rule (Eq. (3))

$$VV_{SAFA} = \frac{1}{r}\sum_{t=1}^{r}\mathrm{var}(V_t), \tag{10}$$

where V_t is the version distribution of trained clients at round t, i.e., $V_t = \{v_1, v_2, \ldots, v_m\}$.

We change *lag tolerance* (i.e., τ) from 1 to 10 and set up several groups of FL tests running a regression task on the Boston Housing dataset. We set the maximum number of global rounds to 100. Apart from the best loss achieved (i.e., the minimum loss by the global model in 100 rounds), we also present the statistical results in the metrics including EUR, SR and VV.

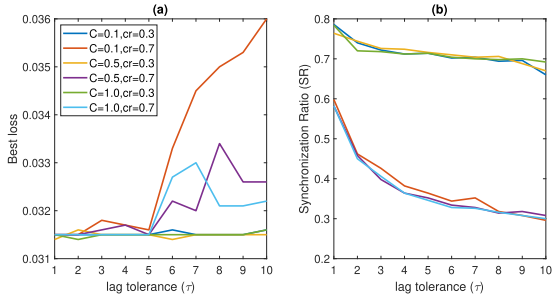


Fig. 3. (a) Best loss achieved by the global model and (b) the synchronization ratio over the federated optimization with SAFA protocol under different lag tolerance settings.

Fig. 3a draws the best loss of the global model in the FL environment where we set the selection fraction C to 0.1, 0.5 and 1.0, and set the expectation of client crash probability cr to 0.3 and 0.7, respectively. Fig. 3b shows the resulting synchronization ratio (SR). Apparently small values of *lag tolerance* show a clear advantage in terms of loss. However, the overhead of communication (revealed by SR) is relatively large in the case where τ is set too small (e.g., 1, 2 or 3). This is expected because more clients will become deprecated and be forced to synchronize when we are less tolerant to the stragglers and stale models.

There are multiple factors that can affect the best global model we can obtain in federated learning. We analyze it by observing the effective update ratio (EUR) and the variance of version (VV) under different FL settings – we argue that they are two important metrics that well reflect the quality of the aggregation step, which is vital for the accuracy of the global model. From Fig. 4a we can see EUR basically remains level as the lag tolerance changes, and that EUR depends on both the client fraction C and the client crash probability cr. When cr is low (e.g., $cr = 0.3$), EUR is slightly above the percentage quota of the clients specified by C, which is because of the contribution by undrafted clients. In the case of a high crash rate (e.g., $cr = 0.7$), EUR is restricted at a low level as it is impossible to be higher than $\mathbb{E}(|M - K|)$, which in theory is equal to $1 - cr$, i.e., the portion of clients with successfully committed updates. In addition, the plot of Version Variance in Fig. 4b reveals part of the reason why the quality of the global model degrades when *lag tolerance* is set too large (see Fig. 3a). In general VV increases if we make SAFA more tolerant to the stragglers (i.e., a larger value of τ). It can be further observed from Fig. 4b that as τ increases, VV goes up at a much slower rate

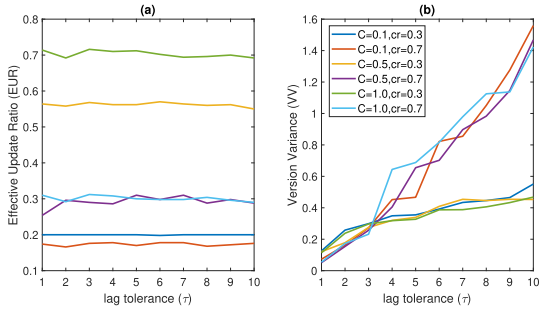


Fig. 4. (a) Effective Update Ratio (EUR) and (b) Version Variance (VV) over the federated optimization with SAFA protocol under different lag tolerance settings.

in relatively stable FL settings (e.g., $cr = 0.3$) than in the extreme settings (e.g., $cr = 0.7$). Combining Figs. 3a and 4b, we can see a clear correlation between VV and the quality of global model especially in an unstable environment where the clients disengage frequently.

Based on the observations, we find that a moderate *lag tolerance* can largely restrain the loss of global model below a desired level and avoid the high communication cost (indicated by SR) in sending out the global model. Therefore we suggest setting *lag tolerance* to 5 rounds in general.

3.5 Bias Analysis

In this section, we theoretically analyze the bias in client selection introduced by the discrepancy of performance and reliability between clients. Here the bias between two clients (e.g., clients A and B) refers to the ratio of client A's chance of contributing to the global model to client B's chance. It is worth mentioning that FedAvg also incurs bias (even though it uses random selection before the training starts) because the clients drop or opt out with different frequencies.

In the analysis, we consider an extreme case which represents the worst bias between the clients. In this case, clients A and B are assumed to the most and least powerful clients, respectively. Namely, clients A and B yield the shortest and longest local training time, respectively. Further, we assume they have the probabilities of dropping out in any round of training, which are denoted by cr_A and cr_B. For the entire set of clients, we assume an overall crash ratio, denoted by R, which is the expected proportion of clients that drop out in a FL round. After r rounds of training, the bias between clients A and B can be represented by

$$bias^{(r)} = \frac{P^{(r)}(A)}{P^{(r)}(B)}, \tag{11}$$

where $P^{(r)}(A)$ (or $P^{(r)}(B)$) denotes the probability that the local update of client A (or B) is successfully aggregated in the global aggregation step in round r.

We first analyze the bias generated by FedAvg (see Eq. (12)), which selects clients at the beginning of a round and the server will wait for all these selected clients to submit local updates. The local update of a selected client will always be aggregated in this round unless this client crashed. Therefore, the bias in FedAvg only depends on the clients' crash rates, which can be modeled by

$$bias^{(r)}_{FedAvg} = \frac{1 - cr_A}{1 - cr_B}. \tag{12}$$

In SAFA, the situation is different. C percentage of the clients are selected from all clients that committed their local updates at the end of this round. The bias in SAFA not only depends on the crash rate, but also on the performance of the clients. A more powerful client can complete their local training faster and therefore its local update has a higher chance to be used in a round. For example, when C percentage of the clients submit their local updates, the server will be able to finish its client selection stage and consequently the clients who fail to finish/submit before that will miss that round.

There are two possible cases where a local update can be used in the current round: i) When a client is selected by the server, its local update will be directly applied in the current round. We denote the probability of this case by $P^{(r)}_D(A)$. ii) The local update generated by an undrafted client in last round also has the chance to be used in the current round through the bypass scheme. The probability that this case occurs is denoted by $P^{(r)}_S(A)$. Therefore, $P^{(r)}(A)$ can be calculated by summing up $P^{(r)}_D(A)$ and $P^{(r)}_S(A)$. $P^{(r)}(B)$ is decomposed similarly.

Due to the space limitation, we only present the final expressions of $P^{(r)}(A)$ and $P^{(r)}(B)$ below in this section. The detailed derivation steps can be found in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TC.2020.2994391.

First, we need to consider three cases of client selection in SAFA given selection fraction C and crash ratio R:

case 1 $\Leftrightarrow C \geq 1 - R$
case 2 $\Leftrightarrow (1 - C)(1 - R) \leq C < 1 - R$
case 3 $\Leftrightarrow C < (1 - C)(1 - R)$.

Literally, case 1 represents a deficit in client selection (i.e., too many crashes to fulfill the pick percentage C). Case 3 means that we can meet the selection ratio C by only selecting the arrived updates from clients not selected last round since they are prioritized by SAFA. Case 2 stands between cases 1 and 3. Namely, we meet the selection ratio C by selecting the prioritized (i.e., last-round undrafted or crashed) clients first and then other clients who also committed their local updates in this round. Considering these cases we have the following proposition:

Proposition 1. *The probabilities $P^{(r)}(A)$ and $P^{(r)}(B)$ can be formulated respectively by Eqs. (13) and (14) given $r > 1$:*

$$P^{(r)}(A) = \begin{cases} 1 - cr_A & \text{if case 1,} \\ 1 - cr_A & \text{if case 2,} \\ \sigma_A^{(r-1)} - cr_A^2 & \text{otherwise} \end{cases} \tag{13}$$

$$P^{(r)}(B) = \begin{cases} 1 - cr_B & \text{if case 1,} \\ \sigma_B^{(r-1)} - cr_B^2 & \text{if case 2,} \\ 1 - cr_B & \text{otherwise} \end{cases} \tag{14}$$

where $\sigma^{(k)}_A = 1 - P^{(k)}_D(A)$ and $\sigma^{(k)}_B = 1 - P^{(k)}_D(B)$. The proof of the proposition is detailed in Appendix A, available in the online supplemental material. Combining the expressions of $P^{(r)}_D(A)$ and $P^{(r)}_D(B)$ (see Eqs. (22) and (24) in Appendix A, available in the online supplemental material) with proper reduction, we can derive $\sigma^{(k)}_A$ and $\sigma^{(k)}_B$

$$\begin{cases} \sigma^{(k)}_A = \frac{2cr_A - (cr_A - 1)^{k+1} - 3}{cr_A - 2} \\ \sigma^{(k)}_B = \frac{2cr_B - (cr_B - 1)^{k+1} - 3}{cr_B - 2} \end{cases}. \tag{15}$$

Therefore, combining Eqs. (13), (14) and the definition of bias, we can derive the bias introduced by SAFA in round r ($r > 1$) as follows:

$$bias^{(r)}_{SAFA} = \begin{cases} \frac{1 - cr_A}{1 - cr_B} & \text{if case 1,} \\ \frac{1 - cr_A}{\sigma_B^{(r-1)} - cr_B^2} & \text{if case 2,} \\ \frac{\sigma_A^{(r-1)} - cr_A^2}{1 - cr_B} & \text{otherwise} \end{cases} \tag{16}$$

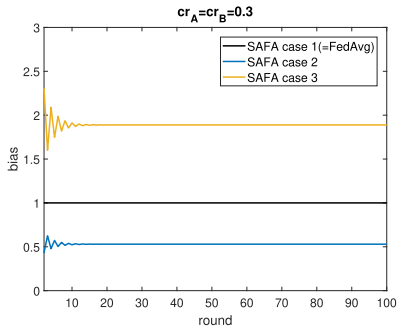


Fig. 5. The bias incurred by FedAvg and SAFA (in the circumstances of three different cases) as a function of federated round index. Here both clients A and B have the same crash rate of 0.3, and the results are similar when setting different cr_A and cr_B according to our experiments.

Fig. 5 visualizes the bias of FedAvg and SAFA as a function of round index r. In case 1 where all local updates committed by the clients are aggregated, we have a fixed bias of $\frac{1-cr_A}{1-cr_B}$, which is the same as FedAvg. In case 2, client B, as the slowest one, will be picked (once it has committed) by the server as long as it was undrafted or crashed in the previous round, which effectively reduces the bias to a level below that of FedAvg. As for case 3, the quota (decided by C) will be fulfilled only with last-round undrafted or crashed clients. Assuming both A and B missed last round, client B is disadvantageous because the server is likely to end the round before B finishes training when the fraction C has been fulfilled by other faster clients (including client A). In all these cases, the bias between A and B converges after a few rounds once FL starts.

4 Experimental Evaluation

4.1 Experiment Setup

We conducted extensive experiments to evaluate the effectiveness of the SAFA protocol on three typical machine learning tasks. *Task 1* is to fit a regression model on the public Boston Housing dataset,[1] which is available in public repositories. *Task 2* is to learn a handwritten digit image classification model implemented using a convolutional neural network (CNN), which is comprised of two 5x5 convolution layers (the first one with 20 channels and the second with 50 channels) with 2x2 max pooling, a fully-connected layer with ReLu as the activation function, and a final softmax output layer. This light-weight CNN is suitable for end devices with small memory size and also adopted in the experiment by [3]. *Task 3* is to learn a classification model for detecting network intrusion given the TCP dump data. For this task we extract the TCP-connection examples from the KDD Cup'99 dataset[2] and use Support Vector Machine (SVM) as the classification model.

We set up separate environments for these three learning tasks to investigate the performance of our protocol in different FL settings. To simulate the unreliability of clients, we set a crash probability (cr) in each run of test and assume each client has the equal chance cr to drop out in any round

1. Online. [Available]: https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html

2. Online. [Available]: https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

TABLE 2
Experimental Setup for Federated Learning

parameter	symbol	Task 1	Task 2	Task 3
dataset	D	Boston	MNIST	KDDCup99
# of features	d	13	28x28	35
model	w	Regression	CNN	SVM
dataset size	n	506	70k	186k
# of clients	m	5	100	500
max # of rounds	R	100	50	100
# of local epochs	E	3	5	5
mini-batch size	B	5	40	100
learning rate	lr	1e-4	1e-3	1e-2

of federated training. For a given task, we use identical local training settings (e.g., mini-batch size) for all the clients and use identical global settings (e.g., the maximum number of rounds and round time limit) for each protocol. The details of the experiment setup is shown in Table 2.

To simulate data imbalance and the heterogeneity in end devices, we assume the size of data partitions (i.e., local data size) follows the Gaussian distribution $\mathcal{N}(\mu, 0.3\mu)$ where $\mu = n/m$, and assume clients' performance follows the exponential distribution with $\lambda = 1.0$. Here we define the performance of a client as the number of batches it can process per second in training. End devices (i.e., clients) may be unreliable and crash occasionally by a probability of ρ_k. In the experiment we assume clients crash independently with the same probability in any federated round and set ρ_k to be cr, i.e., $\rho_k = cr, k = 1, 2, \ldots, m$.

For comparison, we also implemented FedAvg [3], FedCS and a fully local training process as the baselines. FedCS [20] is a refined FL protocol that has to estimate the speed that clients work and filters out some slow clients proactively (at the stage of client selection) to improve the overall efficiency of FL. The fully local protocol never performs the global aggregation until the end of the final round.

4.2 Results

In this section we present the results of our experiments and discuss the evaluated FL protocols in terms of the quality of the obtained global model (shown in Figs. 6, 7 and 8, with more details in Tables 10, 12 and 14, available in the online

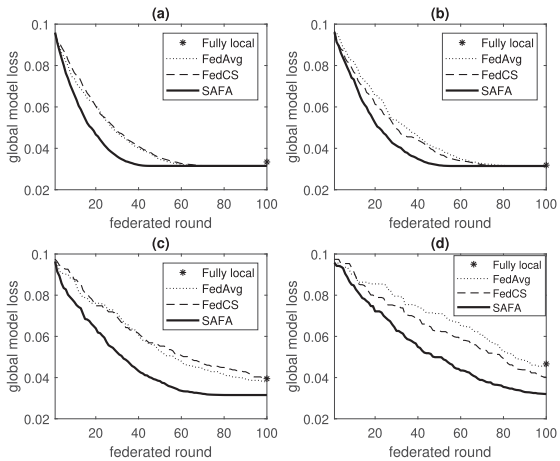


Fig. 6. The loss trace of the global model as the FL process progresses on Task 1 where the client fraction is set to 0.3 and the crash probability is set to 0.1, 0.3 ,0.5, and 0.7 for the four sub-figures (a)–(d), respectively.

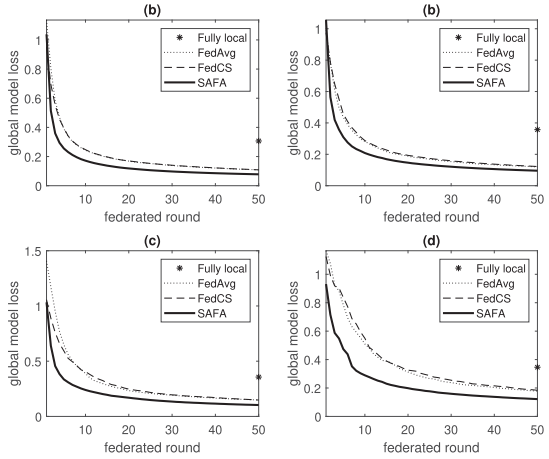


Fig. 7. The loss trace of the global model as the FL process progresses on Task 2 where the client fraction is set to 0.3 and the crash probability is set to 0.1, 0.3 ,0.5, and 0.7 for the four sub-figures (a)–(d), respectively.

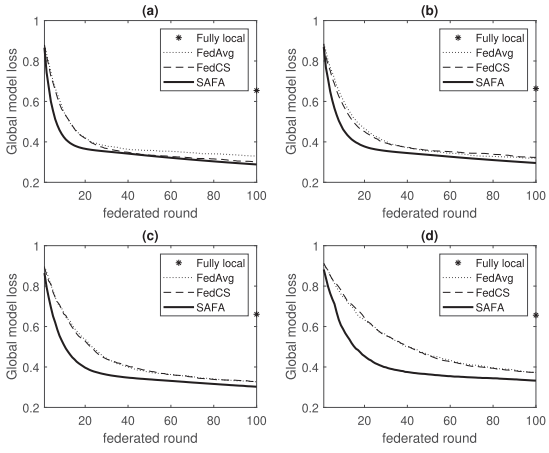


Fig. 8. The loss trace of the global model as the FL process progresses on Task 3, where the client fraction is set to 0.3 and the crash probability is set to 0.1, 0.3 ,0.5, and 0.7 for the four sub-figures (a)–(d), respectively.

TABLE 3
Formulating Accuracy of the Global Model for the Three Tasks

ML task	accuracy formulation
Task 1: regression	$acc = 1 - \frac{1}{n}\sum_{i=1}^{n} \frac{\lvert y_i - \hat{y}_i \rvert}{\max(y_i, \hat{y}_i)}$
Task 2: CNN	$acc = \frac{1}{n}\sum_{i=1}^{n} \phi(y_i, \hat{y}_i)$
Task 3: SVM	$acc = \frac{1}{n}\sum_{i=1}^{n} \max(0, \text{sign}(y_i \cdot \hat{y}_i))$

TABLE 4
Average Length of a Federated Round in Secs on Task 1 Wherein Each Protocol was Tested With Varying Selection Fraction Under Different Environment Settings

Avg. round length (Task 1: regression)					
			FedAvg		
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	316.22	489.37	586.90	731.12	808.59
0.3	429.63	652.39	641.40	736.53	832.02
0.5	372.43	495.37	475.14	621.91	676.41
0.7	354.34	405.86	593.10	728.25	661.67
			FedCS		
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	207.50	487.47	564.20	656.49	786.96
0.3	336.97	519.58	651.23	401.95	832.02
0.5	186.51	221.46	467.98	621.91	676.41
0.7	195.09	398.81	584.68	393.09	661.67
			SAFA		
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	149.69	389.44	540.41	606.48	734.40
0.3	202.44	430.68	583.22	371.77	699.23
0.5	169.33	215.66	408.85	510.85	508.23
0.7	161.81	293.09	402.18	411.06	379.29

Round time limit is set to 830s considering the client performance and data distribution.

TABLE 5
Average Model Distribution Overhead (Unit: Seconds) on Task 1

Avg. T_{dist} (Task 1: regression)					
			FedAvg		
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	0.40	0.81	1.21	1.62	2.02
0.3	0.40	0.81	1.21	1.62	2.02
0.5	0.40	0.81	1.21	1.62	2.02
0.7	0.40	0.81	1.21	1.62	2.02
			FedCS		
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	0.33	0.81	1.21	1.62	2.02
0.3	0.40	0.81	1.21	1.31	2.02
0.5	0.33	0.64	1.21	1.62	2.02
0.7	0.33	0.81	1.21	1.29	2.02
			SAFA		
cr	$C = 0.1$	$C = 0.3$	$C = 0.5$	$C = 0.7$	$C = 1.0$
0.1	1.84	1.83	1.80	1.84	1.81
0.3	1.49	1.46	1.43	1.40	1.41
0.5	1.00	1.07	0.96	1.05	1.02
0.7	0.76	0.69	0.77	0.75	0.74

supplemental material) as well as holistic metrics including round efficiency (summarized in Tables 4, 6 and 8), communication overheads (in Tables 5, 7 and 9) and local resource utilization (Tables 11, 13 and 15, available in the online supplemental material).

A main objective of our work is to boost the round efficiency (i.e., reducing the average length of a federated round), the convergence rate and the resulting accuracy (of the global model). In our experiments, we measure the length of a federated round by considering both local training time and communication overheads, which is captured by Eq. (17)

$$T = \min\{T_{lim},\, T_{dist} + \max_k\{T_k^{down} + T_k^{up} + T_k^{train}\}\}, \quad (17)$$

where T_{lim} is the preset upper limit of round length. T_k^{train}, T_k^{down} and T_k^{up} denote local training time, model download and upload time for client k, respectively. T_k^{down} and T_k^{up} depend on model size and device bandwidth. Using a local network setting similar to that in [20], we assign a stable bandwidth of 1.40 Mbps to each client. For client k, its local training time (i.e., T_k^{train}) is determined using Eq. (18):

$$T_k^{train} = \frac{|B_k| \cdot E}{s_k}, \quad (18)$$

TABLE 6
Average Length of a Federated Round in Secs on Task 2 Wherein Each Protocol was Tested With Varying Selection Fraction Under Different Environment Settings

Avg. round length (Task 2: CNN)					
FedAvg					
cr	$C=0.1$	$C=0.3$	$C=0.5$	$C=0.7$	$C=1.0$
0.1	3402.55	5557.25	5610.20	5614.28	5620.40
0.3	5410.97	5606.12	5610.20	5614.28	5620.40
0.5	5602.04	5606.12	5610.20	5614.28	5620.40
0.7	5602.04	5606.12	5610.20	5614.28	5620.40
FedCS					
cr	$C=0.1$	$C=0.3$	$C=0.5$	$C=0.7$	$C=1.0$
0.1	1487.96	2133.02	3668.70	1871.65	1982.91
0.3	1261.59	1542.61	3132.86	2349.46	5395.54
0.5	1273.37	1642.59	3025.75	2876.63	3162.02
0.7	1253.74	1969.28	2180.46	4344.88	2530.01
SAFA					
cr	$C=0.1$	$C=0.3$	$C=0.5$	$C=0.7$	$C=1.0$
0.1	198.28	315.33	3703.81	1708.93	1947.90
0.3	206.88	368.01	2691.25	1899.23	2149.23
0.5	203.48	800.64	2573.60	2727.25	2186.67
0.7	241.86	1893.14	1877.30	2619.79	2340.80

Round time limit is set to 5600s considering the client performance and data distribution.

TABLE 8
Average Length of a Federated Round in Secs on Task 3 Wherein Each Protocol was Tested With Varying Selection Fraction Under Different Environment Settings

Avg. round length (Task 3: SVM)					
FedAvg					
cr	$C=0.1$	$C=0.3$	$C=0.5$	$C=0.7$	$C=1.0$
0.1	1640.20	1680.60	1721.00	1761.40	1822.00
0.3	1640.20	1680.60	1721.00	1761.40	1822.00
0.5	1640.20	1680.60	1721.00	1761.40	1822.00
0.7	1640.20	1680.60	1721.00	1761.40	1822.00
FedCS					
cr	$C=0.1$	$C=0.3$	$C=0.5$	$C=0.7$	$C=1.0$
0.1	788.75	1319.17	1607.42	1539.14	1802.09
0.3	685.26	1216.12	1521.82	1617.97	1775.50
0.5	714.73	1229.87	1371.03	1605.23	1821.60
0.7	754.52	1190.44	1526.23	1573.42	1731.65
SAFA					
cr	$C=0.1$	$C=0.3$	$C=0.5$	$C=0.7$	$C=1.0$
0.1	310.70	353.98	1419.29	1514.38	1802.15
0.3	274.03	330.32	1499.79	1559.50	1762.51
0.5	242.93	398.27	1317.91	1476.14	1724.52
0.7	212.52	1187.96	1313.99	1223.72	1690.61

Round time limit is set to 1620s considering the client performance and data distribution.

where E is the number of local epochs and $|B_k|$ is the number of batches on device k ($|B_k|$ depends on the size of its local data partition and the preset batch size). In the experiment, a client's performance is defined as the number of batches the client is able to process per second. s_k denotes the performance of client k. We assume that clients' performance follows the exponential distribution with $\lambda = 1.0$.

For a federated round, T_{dist} denotes the server-side overhead for distributing the global model to the end devices. In this paper we assume the server can fully utilize its bandwidth to send models in parallel via intermediate network elements [19] to the clients. Thus T_{dist} depends on the number of model copies to distribute (denoted by m_{sync}) and the communication bandwidth of the server (denoted by bw). T_{dist} is formulated in Eq. (19). Given a FL protocol, T_{dist} of a round is closely correlated to its Synchronization Ratio. The increase in SR indicates a higher average communication cost at the stage of model distribution

$$T_{dist} = \frac{m_{sync} \cdot model\_size}{bw}, \tag{19}$$

TABLE 7
Average Model Distribution Overhead (Unit: Seconds) on Task 2

Avg. T_{dist} (Task 2: CNN)					
FedAvg					
cr	$C=0.1$	$C=0.3$	$C=0.5$	$C=0.7$	$C=1.0$
0.1	2.04	6.12	10.20	14.28	20.40
0.3	2.04	6.12	10.20	14.28	20.40
0.5	2.04	6.12	10.20	14.28	20.40
0.7	2.04	6.12	10.20	14.28	20.40
FedCS					
cr	$C=0.1$	$C=0.3$	$C=0.5$	$C=0.7$	$C=1.0$
0.1	2.04	6.12	10.20	14.14	20.40
0.3	2.02	6.05	10.20	14.13	20.40
0.5	2.04	6.06	10.20	14.28	20.20
0.7	2.04	6.12	10.11	14.28	20.20
SAFA					
cr	$C=0.1$	$C=0.3$	$C=0.5$	$C=0.7$	$C=1.0$
0.1	18.27	18.45	18.26	18.47	18.38
0.3	14.45	14.65	14.48	14.54	14.69
0.5	10.89	10.51	10.70	10.84	10.58
0.7	7.17	7.23	7.55	7.21	7.41

TABLE 9
Average Model Distribution Overhead (in Seconds) on Task 3

Avg. T_{dist} (Task 3: SVM)					
FedAvg					
cr	$C=0.1$	$C=0.3$	$C=0.5$	$C=0.7$	$C=1.0$
0.1	20.20	60.60	101.00	141.40	202.00
0.3	20.20	60.60	101.00	141.40	202.00
0.5	20.20	60.60	101.00	141.40	202.00
0.7	20.20	60.60	101.00	141.40	202.00
FedCS					
cr	$C=0.1$	$C=0.3$	$C=0.5$	$C=0.7$	$C=1.0$
0.1	20.20	60.48	100.78	140.79	201.60
0.3	20.11	60.60	100.81	141.09	201.60
0.5	20.13	60.60	100.85	140.84	201.60
0.7	20.20	60.60	100.61	141.14	201.19
SAFA					
cr	$C=0.1$	$C=0.3$	$C=0.5$	$C=0.7$	$C=1.0$
0.1	181.95	182.32	181.49	181.84	182.15
0.3	142.89	141.91	141.95	142.50	142.81
0.5	104.38	104.56	105.34	104.59	104.52
0.7	70.62	70.63	70.55	70.05	70.61

where the server bandwidth bw is set to 10 Gbps in our experiment considering the prevailing 10-Gigabit Ethernet connection. Models are usually compressed before transmission. We use 10 MB as the model size following the result presented in [26].

For different machine learning models, we define their accuracy in different ways, as shown in Table 3. In the table, y and $\hat{y}$ denote the label and the output of the model, respectively. The function $\phi(\cdot)$ returns 1 if $\hat{y}$ matches y, otherwise it returns 0.

Task 1: Regression

In this task, we aim to learn a regression model on a small group of clients to predict the median value of a house in the area of Boston Mass. Input features include 13 properties about the estate such as average number of rooms per dwelling and crime rate. In this experiment, we ran FL with every candidate protocol (i.e., SAFA, FedAvg, FedCS and fully local training) and compare their effectiveness in terms of the achieved accuracy of the global model, round efficiency and communication overhead.

It can be seen from Fig. 6 and Table 10 (in Appendix B, available in the online supplemental material) that SAFA significantly improves the convergence rate as well as the best accuracy achieved by the global regression model, especially under settings of unstable environments (i.e., $cr \geq 0.5$). This is mainly attributed to our staleness-tolerant mechanism. Another advantage of the tolerance to stragglers is the preservation of local training results. We use the metric *Futility Percentage* to measure the percentage of local progress that is wasted due to the model synchronization forced by the server (FedAvg and FedCS force the selected clients to overwrite its local model with the latest global model). Results of SR and futility percentage in Table 11, available in the online supplemental material, show that the wasted training progress is reduced by SAFA effectively.

As shown in Tables 4 and 5, there is not much difference in average round length and model distribution overhead due to the very limited number of devices used to run task 1. But we still observed notable efficiency boost and convergence speedup by SAFA under the circumstance where the selection fraction C is very small. With C set to 0.1, SAFA halves the time required to finish a federated round compared to FedAvg.

Task 2: CNN

We divided the MNIST dataset into m partitions of which the sizes are random variables (following Gaussian distribution). The CNN models with randomly initialized weights are created on 100 clients and we again tested Fully local, FedAvg, FedCS and SAFA under a variety of FL settings.

As a result, the Fully Local protocol can finish with an accuracy around 90 percent on this classification task with the CNN model, while FedAvg, FedCS and SAFA raise that to $96.0\% \sim 98.0\%$ (Table 12 in Appendix B, available in the online supplemental material). SAFA shows a significant advantage in round efficiency (see Table 6) - it is able to achieve up to $27\times$ and $6\times$ speed-up compared to FedAvg and FedCS in an unreliable environment where clients frequently opt/drop out and only a small fraction (i.e., $C = 0.1$ or 0.3) of them are allowed to participate in a round.

The average T_{dist} for SAFA mainly depends on client crash probability (see Table 7), and it remains at a low level with $cr \geq 0.5$. In the case where devices are more reliable in local training (i.e., $cr < 0.5$), SAFA embraces a greater number of updates and results in a slightly higher cost (of tens of seconds) during the stage of model distribution, but the overhead is still acceptable considering the overall length of a federated round (which could last thousands of seconds, see Table 6).

Task 3: SVM

For this task we use a relatively large data set containing 186,480 TCP dump records including several types of network intrusions. The target is to learn a global SVM model to recognize malicious connections and normal connections. We dispersed the dataset onto 500 clients to perform FL with SAFA and other existing training protocols.

Table 14 (in Appendix B, available in the online supplemental material) shows that FedAvg, FedCS and SAFA can produce very accurate global models (with the classification accuracy of over 99 percent) after convergence. SAFA could incur higher overhead in model distribution (as the SR is larger for SAFA, see Tables 9 and 15 in some cases). Nevertheless, SAFA still significantly outperforms FedAvg and FedCS by $7.7\times$ and $3.7\times$, respectively, in average round length (see Table 8). Its advantage decreases as more clients are set to engage in training but it is still the most efficient protocol. In contrast to FedAvg and FedCS, SAFA capitalizes the contribution from straggling clients effectively, leading to a very small futility percentage (below 4 percent, see Table 15 in Appendix B, available in the online supplemental material) on this task, which means that the majority of local training progresses make contribution to the convergence of the final global model, even in a very unreliable environment.

4.3 Discussion

The experimental results with several tasks including regression and classification demonstrate the effectiveness of applying our semi-asynchronous protocol to FL with unreliable clients. The improvement achieved by SAFA lies in three-fold: i) faster convergence of the global model and a higher accuracy achieved, ii) significant reduction in average round length, and iii) increased utilization of local training progress made by the stragglers. A few interesting phenomena were also observed in our experiments. First and foremost, we find that increasing the client fraction C does not always improve the quality of the global model. For example, a reasonably high accuracy is obtained by setting C to 0.3 or 0.5 (instead of 1.0) in task 2 in the case of a low crash probability. This in some ways infers that involving more clients each round is not always beneficial (or has very limited benefit). In addition, we notice that fully local training without round-wise aggregation is in some cases able to produce a reasonably good model, e.g., in the cases of Task 1 with $C = 0.3$ and Task 3 with $C = 0.1$ and $cr = 0.7$. Also, we find that the synchronous FL protocol FedAvg can produce a global model slightly better than our solution in the case of $C = 1.0$, i.e., trying to involve all clients in every round. This advantage is probably brought by the feature that pure synchronization can avoid the negative effect from stale models, which amplifies as a larger fraction of clients get involved. However, it is practically unrealistic to set a big C for FL because communication could be expensive while the enhancement of the resulting accuracy is very limited (see

Tables 10, 12 and 14 in Appendix B, available in the online supplemental material).

5 Conclusion

Aiming at improving the efficiency of federated learning with unreliable end devices, we propose a semi-asynchronous protocol which incorporates a novel client selection algorithm decoupling the central server and the selected clients for a reduction of average round time as well as a lag-tolerant mechanism in model distribution for tackling the tradeoff between faster convergence and lower communication overhead. We also analyze the upper bound of the bias introduced by using SAFA in FL. The results of experimental evaluation on three typical machine learning tasks show that our protocol effectively enhances the round efficiency of federated optimization process, improves the quality of the global model and reduces local resource wastage at a relatively low cost of communication.

Considering the subtle correlation between local models and the global model, we plan to look into the balance between generating the best local models for end devices and obtaining an optimal global model in the central server. As another part of future work, we are also going to investigate how to further improve federated learning using model parallelism and compression.

Acknowledgments

This work was supported in part by Worldwide Byte Security Information Technology Company ltd., in part by National Natural Science Foundation of China under Grant 61772205, in part by Guangzhou Development Zone Science and Technology under Grant 2018GH17, in part by Major Program and of Guangdong Basic and Applied Research under Grant 2019B030302002, in part by Guangdong project under Grant 2017B030314073 and Grant 2018B030325002, in part by the EPSRC Centre for Doctoral Training in Urban Science under EPSRC Grant EP/L016400/1, in part by the Alan Turing Institute under EPSRC Grant EP/N510129/1 and Grant PETRAS, and in part by the National Center of Excellence for IoT Systems Cybersecurity under Grant EP/S035362/1.

References

[1] 3 AI Trends for Enterprise Computing. 2017. [Online] Available: https://www.gartner.com/smarterwithgartner/3-ai-trends-for-enterprise-computing/

[2] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," in *Proc. 29th Conf. Neural Inf. Process. Syst.*, 2016.

[3] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artif. Intell. Statist.*, 2017, pp. 1273–1282.

[4] K. Bonawitz *et al.*, "Towards federated learning at scale: System design," in *Proc. Conf. Syst. Mach. Learn.*, 2019.

[5] J. Chen, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous SGD," in *Proc. Int. Conf. Learn. Representations Workshop Track*, 2016.

[6] I. M. Baytas, M. Yan, A. K. Jain, and J. Zhou, "Asynchronous multi-task learning," in *Proc. Int. Conf. Data Mining*, 2016, pp. 11–20.

[7] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous SGD," 2016, *arXiv:1604.00981*.

[8] V. Smith, C. K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 4424–4434.

[9] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," *arXiv: 1903.03934*.

[10] M. R. Sprague *et al.*, "Asynchronous federated learning for geospatial applications," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discov. Databases*, 2018, pp. 21–28.

[11] E. Li, Z. Zhou, and X. Chen, "Edge intelligence: On-demand deep learning model co-inference with device-edge synergy," in *Proc. Workshop Mobile Edge Commun.*, 2018, pp. 31–36.

[12] S. Deng, H. Zhao, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge intelligence: The confluence of edge computing and artificial intelligence," to be published, doi: 10.1109/JIOT.2020.2984887.

[13] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1738–1762, Aug. 2019.

[14] J. Wu, W. Huang, J. Huang, and T. Zhang, "Error compensated quantized SGD and its applications to large-scale distributed optimization," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 5321–5329.

[15] P. Goyal *et al.*, "Accurate, large minibatch SGD: Training imagenet in 1 hour," 2017, *arXiv: 1706.02677*.

[16] S. Zheng *et al.*, "Asynchronous stochastic gradient descent with delay compensation," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 4120–4129.

[17] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: communication-efficient SGD via gradient quantization and encoding," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 1709–1720.

[18] S. Wang *et al.*, "When edge meets learning: Adaptive control for resource-constrained distributed machine learning," in *Proc. IEEE Conf. Comput. Commun.*, 2018 pp. 63–71.

[19] S. Wang *et al.*, "Adaptive federated learning in resource constrained edge computing systems," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1205–1221, Jun. 2019.

[20] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *Proc. IEEE Int. Conf. Commun.*, 2019, pp. 1–7.

[21] J. Chen, K. Li, Q. Deng, K. Li, and S. Y. Philip, "Distributed deep learning model for intelligent video surveillance systems with edge computing," *IEEE Trans. Ind. Informat.*, to be published, doi: 10.1109/TII.2019.2909473.

[22] A. Hard *et al.*, "Federated learning for mobile keyboard prediction," 2018, *arXiv: 1811.03604*.

[23] X. Lian, Y. Huang, Y. Li, and J. Liu, "Asynchronous parallel stochastic gradient for nonconvex optimization," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 2737–2745.

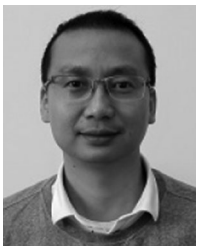
[24] S. Dutta, G. Joshi, S. Ghosh, P. Dube, and P. Nagpurkar, "Slow and stale gradients can win the race: Error-runtime trade-offs in distributed SGD," 2018, *arXiv: 1803.01113*.

[25] W. Zhang, S. Gupta, X. Lian, and J. Liu, "Staleness-aware Async-SGD for distributed deep learning," in *Proc. 25th Int. Joint Conf. Artif. Intell.*, 2015, pp. 2350–2356.

[26] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. Int. Conf. Learn. Representations*, 2016.

[27] P. Kairouz *et al.*, "Advances and open problems in federated learning," 2019, *arXiv: 1912.04977*.

Wentai Wu (Student Member, IEEE) received the bachelor's and master's degrees in computer science from the South China University of Technology, Guangzhou, China, in 2015 and 2018, respectively. Currently, he is working toward the PhD degree with the Department of Computer Science, University of Warwick, Coventry, United Kingdom supervised by Dr. Ligang. His research interests mainly include parallel and distributed computing, distributed machine learning, and energy-efficient computing.



Ligang He (Member, IEEE) received the PhD degree in computer science from the University of Warwick, Coventry, United Kingdom, and worked as a post-doctoral researcher with the University of Cambridge, United Kingdom. From 2006, he worked with the Department of Computer Science, University of Warwick as an assistant professor, and then became an associate professor. He is currently a reader with the Department. His research interests focus on parallel and distributed processing, cluster, grid, and cloud computing. He has published more than 100 papers in international conferences and journals, such as the *IEEE Transactions on Parallel and Distributed Systems*, IPDPS, CCGrid, and MAS-COTS. He has been a co-chair or a member of the program committee for a number of international conferences, and been the reviewers for many international journals, including the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Computers*, etc.

Weiwei Lin received the BS and MS degrees from Nanchang University, Nanchang, China, in 2001 and 2004, respectively, and the PhD degree in computer application from the South China University of Technology, Guangzhou, China, in 2007. Currently, he is a professor with the School of Computer Science and Engineering, South China University of Technology. His research interests include distributed systems, cloud computing, big data computing, and AI application technologies. He has published more than 80 papers in refereed journals and conference proceedings.

Rui Mao received the BS and MS degrees in computer science from the University of Science and Technology of China, Hefei, China, in 1997 and 2000, respectively, and the MS degree in statistics and the PhD degree in computer science from the University of Texas at Austin, Austin, Texas, in 2006 and 2007, respectively. After three years of working with the Oracle USA Corporation, he joined Shenzhen University (SZU), China, in 2010. He is currently an associate professor and an associate dean with the College of Computer Science and Software Engineering, SZU. His research interests include universal data management and analysis in metric space, and high performance computing.

Carsten Maple is a professor of Cyber Systems Engineering in WMG, University of Warwick. He is principal investigator of the NCSC-EPSRC Academic Center of Excellence in Cyber Security Research at the University. He is the Transport & Mobility lead of the PETRAS National Center of Excellence for IoT Systems Cybersecurity. He has published more than 200 peer-reviewed papers and provided evidence and advice to governments and organizations across the world, including being a high-level scientific advisor for cyber security to the European Commission. He is a member of various national and international boards and expert groups and is a fellow of the Alan Turing Institute, the National Centre for Datas Sc pence and AI. He is an executive committee member of the UK-RAS Network and the IoT Security Foundation and an expert group member on automotive security for ENISA, and Interpol.

Stephen Jarvis (Member, IEEE) studied at London, Oxford and Durham Universities before taking his first academic position with the Oxford University Computing Laboratory. He subsequently joined the University of Warwick and was appointed to a personal professorship, in 2009. He acted as director of research in computer science from 2008 to 2013, and was chair of Department from 2013 to 2017. He has published more than 230 academic papers. He is a visiting exchange professor with the New York University and is a non-executive director of the Alan Turing Institute, the UK's National Institute for AI and Data Science. He is presently deputy pro vice chancellor (Research) with the University of Warwick.