Blockchained On-Device Federated Learning

Hyesung Kim, Jihong Park, Mehdi Bennis, and Seong-Lyun Kim

***Abstract*— By leveraging blockchain, this letter proposes a blockchained federated learning (BlockFL) architecture where local learning model updates are exchanged and verified. This enables on-device machine learning without any centralized training data or coordination by utilizing a consensus mechanism in blockchain. Moreover, we analyze an end-to-end latency model of BlockFL and characterize the optimal block generation rate by considering communication, computation, and consensus delays.**

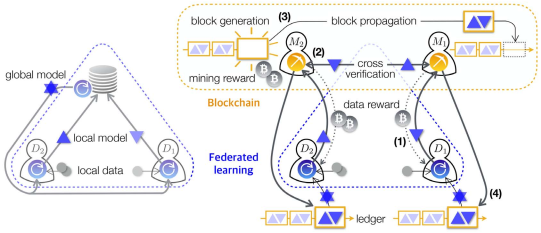
***Index Terms*— On-device machine learning, federated learning, blockchain, latency.**

I. Introduction

FUTURE wireless systems are envisaged to ensure low latency and high reliability anywhere and anytime [1]–[3]. To this end, on-device machine learning is a compelling solution wherein each device stores a high-quality machine learning model and is thereby capable of make decisions, even when it loses connectivity. Training such an on-device machine learning model requires more data samples than each device's local samples, and necessitates sample exchanges with other devices [4]–[6]. In this letter, we tackle the problem of training each device's local model by federating with other devices.

One key challenge is that local data samples are owned by each device. Thus, the exchanges should keep the raw data samples private from other devices. For this purpose, as proposed in Google's federated learning (FL) [4], [5], referred to as *vanilla FL*, each device exchanges its *local model update*, i.e., learning model's weight and gradient parameters, from which the raw data cannot be derived. As illustrated in Fig. 1-a, the vanilla FL's exchange is enabled by the aid of a central server that aggregates and takes an ensemble average of all the local model updates, yielding a *global model update*. Then, each device downloads the global model update, and computes its next local update until the global model training is completed [5]. Due to these exchanges, the vanilla FL's training completion latency might be tens of minutes or more, as demonstrated in Google's keyboard application [7].

The limitation of the vanilla FL operation is two-fold. Firstly, it relies on a single central server, which is vulnerable to the server's malfunction. This incurs inaccurate global model updates distorting all local model updates. Secondly, it does not reward the local devices. A device having a larger number of data samples contributes more to the global training. Without providing compensation, such a device is less willing to federate with the other devices possessing few data samples.



(a) Vanilla FL. (b) Proposed BlockFL.

Fig. 1. An illustration of (a) the vanilla federated learning (FL) [4], [5] and (b) the proposed blockchained FL (BlockFL) architectures.

In order to resolve these pressing issues, by leveraging *blockchain* [8], [9] in lieu of the central server, we propose a *blockchained FL (BlockFL)* architecture, where the blockchain network enables exchanging devices' local model updates while verifying and providing their corresponding rewards. BlockFL overcomes the single point of failure problem and extends the range of its federation to untrustworthy devices in a public network thanks to a validation process of the local training results. Moreover, by providing rewards proportional to the training sample sizes, BlockFL promotes the federation of more devices with a larger number of training samples.

As shown in Fig. 1-b, the logical structure of BlockFL consists of devices and miners. The miners can physically be either randomly selected devices or separate nodes such as network edges (i.e., base stations in cellular networks), which are relatively free from energy constraints in mining process. The operation of BlockFL is summarized as follows: Each device computes and uploads the local model update to its associated miner in the blockchain network; Miners exchange and verify all the local model updates, and then run the Proof-of-Work (PoW) [8]; Once a miner completes the PoW, it generates a block where the verified local model updates are recorded; and finally, the generated block storing the aggregate local model updates is added to a blockchain, also known as distributed ledger, and is downloaded by devices. Each device computes the global model update from the new block.

Note that the global model update of BlockFL is computed locally at each device. A miner's or a device's malfunction does not affect other devices' global model updates. For the sake of these benefits, in contrast to the vanilla FL, BlockFL needs to account for the extra delay incurred by the blockchain network. To address this, the end-to-end latency model of BlockFL is formulated by considering communication, computation, and the PoW delays. The resulting latency is minimized by adjusting the block generation rate, i.e., the PoW difficulty.

Manuscript received May 6, 2019; accepted May 23, 2019. Date of publication June 10, 2019; date of current version June 10, 2020. This work was partly supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea Government (MSIT) (No. 2018-0-00170, Virtual Presence in Moving Objects through 5G), Basic Science Research Foundation of Korea (NRF) grant funded by the Ministry of Science and ICT (NRF-2017R1A2A2A05069810), and the Mobile Edge Intelligence at Scale (ELLIS) Project at the University of Oulu. The associate editor coordinating the review of this letter and approving it for publication was V.-D. Nguyen. *(Corresponding author: Seong-Lyun Kim.)*

H. Kim and S.-L. Kim are with the School of Electrical and Electronic Engineering, Yonsei University, Seoul 03722, South Korea (e-mail: hskim@ramo.yonsei.ac.kr; slkim@ramo.yonsei.ac.kr).

J. Park and M. Bennis are with the Centre for Wireless Communications, University of Oulu, 4500 Oulu, Finland (e-mail: jihong.park@oulu.fi; mehdi.bennis@oulu.fi).

Digital Object Identifier 10.1109/LCOMM.2019.2921755

II. Architecture and Operation

A. FL Operation in BlockFL

The FL under study is operated by a set of devices $\mathcal{D} = \{1, 2, \cdots, N_D\}$ with $|\mathcal{D}| = N_D$. The i-th device D_i owns a set of data samples $\mathcal{S}_i$ with $|\mathcal{S}_i| = N_i$, and trains its local model. The local model updates of the device D_i is uploaded to its associated miner M_j that is uniformly randomly selected out of a set of miners $\mathcal{M} = \{1, 2, \cdots, N_M\}$.

Our distributed model training focuses on solving a regression problem in a parallel manner, considering a set of the entire devices' data samples $\mathcal{S} = \cup_{i=1}^{N_D} \mathcal{S}_i$ with $|\mathcal{S}| = N_S$. The k-th data sample $s_k \in \mathcal{S}$ is given as $s_k = \{x_k, y_k\}$ for a d-dimensional column vector $x_k \in \mathbb{R}^d$ and a scalar value $y_k \in \mathbb{R}$. The objective is to minimize a loss function $f(w)$ for a global weight vector $w \in \mathbb{R}^d$. The loss function $f(w)$ is chosen as the mean squared error: $f(w) = \frac{1}{N_S} \sum_{i=1}^{N_D} \sum_{s_k \in \mathcal{S}_i} f_k(w)$, where $f_k(w) = (x_k^\top w - y_k)^2/2$. Other loss functions under deep neural networks can readily be incorporated, as done in [10].

In order to solve the above problem, following the vanilla FL settings in [4], the model of the device D_i is locally trained via a stochastic variance reduced gradient algorithm [4], and all devices' local model updates are aggregated using a distributed approximate Newton method. For each epoch, the device D_i's local model is updated with the number N_i of iterations. At the t-th local iteration of the ℓ-th epoch, the local weight $w_i^{(t,\ell)} \in \mathbb{R}^d$ is:

$$w_i^{(t,\ell)} = w_i^{(t-1,\ell)} - \frac{\beta}{N_i} \left(\left[\nabla f_k \left(w_i^{(t-1,\ell)} \right) - \nabla f_k \left(w^{(\ell)} \right) \right] + \nabla f \left(w^{(\ell)} \right) \right) \quad (1)$$

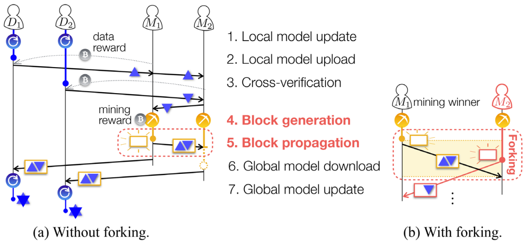
where $\beta > 0$ is a step size, $w^{(\ell)}$ indicates the global weight at the ℓ-th epoch, and $\nabla f(w^{(\ell)}) = 1/N_S \cdot \sum_{i=1}^{N_D} \sum_{s_k \in \mathcal{S}_i} \nabla f_k(w^{(\ell)})$. Let $w_i^{(\ell)}$ denote the local weight after the last local iteration of the ℓ-th epoch, i.e., $w_i^{(\ell)} = w_i^{(N_i,\ell)}$. Then,

$$w^{(\ell)} = w^{(\ell-1)} + \sum_{i=1}^{N_D} \frac{N_i}{N_S} \left(w_i^{(\ell)} - w^{(\ell-1)} \right). \quad (2)$$

In vanilla FL in [4], [5], the device D_i uploads its local model update $(w_i^{(\ell)}, \{\nabla f_k(w^{(\ell)})\}_{s_k \in \mathcal{S}_i})$ to the central server, with the model update size δ_{m} that is identically given for all devices. The global model update $(w^{(\ell)}, \nabla f(w^{(\ell)}))$ is computed by the server. *In BlockFL, the server entity is substituted with a blockchain network* as detailed in the following description.

B. Blockchain Operation in BlockFL

In the BlockFL, the blocks and their verification by the miners in $\mathcal{M}$ are designed so as to exchange the local model updates truthfully through a distributed ledger. Each block in a ledger is divided into its body and header parts [8]. In BlockFL, the body stores the local model updates of the devices in $\mathcal{D}$, i.e., $(w_i^{(\ell)}, \{\nabla f_k(w^{(\ell)})\}_{s_k \in \mathcal{S}_i})$ for the device D_i at the ℓ-th epoch, as well as its local computation time $T_{\mathrm{local},i}^{(\ell)}$ that is discussed at the end of this subsection. The header contains the information of a pointer to the previous block, block generation rate λ, and the output value of the PoW. The size of each block is set as $h + \delta_{\mathrm{m}} N_D$, where h and δ_{m} are the header and model update sizes, respectively. Each miner has a candidate block that is filled with the local model updates from its associated devices and/or other miners. The filling procedure continues until it reaches the block size or a waiting time T_{wait}.



(a) Without forking. (b) With forking.

Fig. 2. The one-epoch operation of BlockFL with and without forking.

Afterwards, following the PoW [8], the miner randomly generates a hash value by changing its input number, i.e., nonce, until the generated hash value becomes smaller than a target value. Once the miner M_1 succeeds in finding the hash value, its candidate block is allowed to be a new block as shown in Fig 2. Here, the block generation rate λ can be controlled by the PoW difficulty, e.g., the lower PoW target hash value, the smaller λ. Due to simpleness and robustness, the PoW is applied to the wireless systems as in [11], [12]. BlockFL can also use other consensus algorithms such as the proof-of-stake (PoS) or Byzantine-fault-tolerance (BFT), which may require more complicated operation and preliminaries to reach consensus among miners.

The generated block is propagated to all other miners. To this end, as done in [8], all the miners receiving the generated block are forced to stop their PoW operations and to add the generated block to their local ledgers. As illustrated in Fig 2, if another miner M_2 succeeds in its block generation within the propagation delay of the firstly generated block, then some miners may mistakenly add this secondly generated block to their local ledgers, known as *forking*. In BlockFL, forking makes some devices apply an incorrect global model update to their next local model updates. Forking frequency increases with λ and the block propagation delay, and its mitigation incurs an extra delay, to be elaborated in Sect. III.

The blockchain network also provides rewards for data samples to the devices and for the verification process to the miners, referred to as *data reward* and *mining reward*, respectively. The data reward of the device D_i is received from its associated miner, and its amount is proportional to the data sample size N_i. When the miner M_j generates a block, its mining reward is earned by the blockchain network, as done in the conventional blockchain structure [8]. The amount of mining reward is proportional to the aggregate data sample size of its all associating devices, namely, $\sum_{i=1}^{N_{M_j}} N_i$ where N_{M_j} denotes the number of devices associated with the miner M_j. It is noted that the BlockFL can further be improved using a reward mechanism, which considers not only the size but also the quality of data sample that affects the accuracy of FL. Untruthful devices may inflate their sample sizes with arbitrary local model updates. Miners verify truthful local updates before storing them by comparing the sample size N_i with its corresponding computation time $T_{\text{local},i}^{(\ell)}$. This can be guaranteed in practice by Intel's software guard extensions, allowing applications to be operated within a protected environment, which is utilized in blockchain technologies [13].

C. One-Epoch BlockFL Operation

As depicted in Fig. 2, the BlockFL operation of the device D_i at the ℓ-th epoch is described by the following seven steps.

1. Local model update: The device D_i computes (1) with the number N_i of iterations.

2. Local model upload: The device D_i uniformly randomly associates with the miner M_i; if $\mathcal{M} = \mathcal{D}$, then M_i is selected from $\mathcal{M}\backslash D_i$. The device uploads the local model updates $(w_i^{(\ell)}, \{\nabla f_k(w^{(\ell)})\}_{s_k \in \mathcal{S}_i})$ and the corresponding local computation time $T_{\text{local},i}^{(\ell)}$ to the associated miner.

3. Cross-verification: Miners broadcast the obtained local model updates. At the same time, the miners verify the received local model updates from their associated devices or the other miners. The verified local model updates are recorded in the miner's candidate block, until its reaching the block size $(h + \delta_{\text{m}} N_D)$ or the maximum waiting time T_{wait}.

4. Block generation: Each miner starts running the PoW until either it finds the nonce or it receives a generated block.

5. Block propagation: Denoting as $M_{\hat{o}} \in \mathcal{M}$ the miner who first finds the nonce. Its candidate block is generated as a new block and broadcasted to all miners. In order to avoid forking, an ACK, including whether forking occurs or not, is transmitted once each miner receives the new block. If a forking event occurs, the operation restarts from Step 1. A miner that generates a new block waits until a predefined maximum block ACK waiting time $T_{\text{a,wait}}$.

6. Global model download: The device D_i downloads the generated block from its associated miner.

7. Global model update: The device D_i locally computes the global model update in (2) by using the aggregate local model updates in the generated block while the parameter N_S in (2) should be replaced with the number of data samples reflected in the block.

The procedure continues until satisfying $|w^{(L)} - w^{(L-1)}| \leq \varepsilon$.

Centralized FL is vulnerable to the server's malfunction that distorts all devices' global models. However, in BlockFL, the global model update is computed locally at each device, which is robust against the malfunction and prevents excessive computational overheads of miners.

III. End-to-End Latency Analysis

We investigate the optimal block generation rate λ^* minimizing learning completion latency T_o, defined as the total time during L epochs at a randomly selected device $D_o \in \mathcal{D}$.

A. One-Epoch BlockFL Latency Model

The ℓ-th epoch latency $T_o^{(\ell)}$ is determined by computation, communication, and block generation delays. First, computation delays are brought by Steps 1 and 7 in Sect. II. Let δ_{d} denote a single data sample's size identical for all data samples. Processing δ_{d} with the clock speed f_c requires δ_{d}/f_c. Local model updating delay $T_{\text{local},o}^{(\ell)}$ in Step 1 is thus given as $T_{\text{local},o}^{(\ell)} = \delta_{\text{d}} N_o / f_c$. Likewise, global model updating delay $T_{\text{global},o}^{(\ell)}$ in Step 7 is evaluated as $T_{\text{global},o}^{(\ell)} = \delta_{\text{m}} N_D / f_c$. Note that δ_d and δ_m change with applications types. Second, communication delays are entailed by Steps 2 and 6 between devices and miners. Measuring the achievable rate under additive white Gaussian noise channels, local model uploading delay $T_{\text{up},o}^{(\ell)}$ in Step 2 is computed as $T_{\text{up},o}^{(\ell)} = \delta_{\text{m}} / [W_{\text{up}} \log_2(1 + \gamma_{\text{up},o})]$, where W_{up} is the uplink bandwidth allocation per device and $\gamma_{\text{up},o}$ is the miner M_o's received signal-to-noise ratio (SNR). The global model downloading delay $T_{\text{dn},o}^{(\ell)}$ in Step 6 is given as $T_{\text{dn},o}^{(\ell)} = (h + \delta_{\text{m}} N_D) / [W_{\text{dn}} \log_2(1 + \gamma_{\text{dn},o})]$, where W_{dn} is the downlink bandwidth per device and $\gamma_{\text{dn},o}$ is the device D_o's SNR.

For Steps 3 and 5, assuming verification processing time is negligible compared to the communication delays, cross-verification delay $T_{\text{cross},o}^{(\ell)}$ in Step 3 is $T_{\text{cross},o}^{(\ell)} = \max\{T_{\text{wait}} - (T_{\text{local},o}^{(\ell)} + T_{\text{up},o}^{(\ell)}), \sum_{M_j \in \mathcal{M}\backslash M_o} \delta_{\text{m}} N_{M_j} / [W_{\text{m}} \log_2(1 + \gamma_{oj})]\}$ under frequency division multiple access (FDMA), where W_{m} is the bandwidth allocation per each miner link and γ_{oj} is the miner M_j's received SNR from the miner M_o. Denoting as $M_{\hat{o}} \in \mathcal{M}$ the miner who first finds nonce, referred to as the mining winner, total block propagation delay $T_{\text{bp},\hat{o}}^{(\ell)}$ in Step 5 is given as $T_{\text{bp},\hat{o}}^{(\ell)} = \max_{M_j \in \mathcal{M}\backslash M_{\hat{o}}} \{t_{\text{bp},j}^{(\ell)}, T_{\text{a,wait}}\}$ under FDMA. The term $t_{\text{bp},j}^{(\ell)} = (h + \delta_{\text{m}} N_D) / [W_{\text{m}} \log_2(1 + \gamma_{\hat{o}j})]$ represents the block propagation delay from the mining winner $M_{\hat{o}}$ to $M_j \in \mathcal{M}\backslash M_{\hat{o}}$, and $\gamma_{\hat{o}j}$ is the miner M_j's received SNR. Lastly, in Step 4, block generation delay $T_{\text{bg},j}^{(\ell)}$ of the miner $M_j \in \mathcal{M}$ follows an exponential distribution with mean $1/\lambda$, as modeled in [9]. The delay of interest is the mining winner $M_{\hat{o}}$'s block generation delay $T_{\text{bg},\hat{o}}^{(\ell)}$. Finally, the ℓ-th epoch latency $T_o^{(\ell)}$ is

$$T_o^{(\ell)} = N_{\text{fork}}^{(\ell)} \left(T_{\text{local},o}^{(\ell)} + T_{\text{up},o}^{(\ell)} + T_{\text{cross},o}^{(\ell)} + T_{\text{bg},\hat{o}}^{(\ell)} + T_{\text{bp},\hat{o}}^{(\ell)} \right) + T_{\text{dn},o}^{(\ell)} + T_{\text{global},o}^{(\ell)}, \quad (3)$$

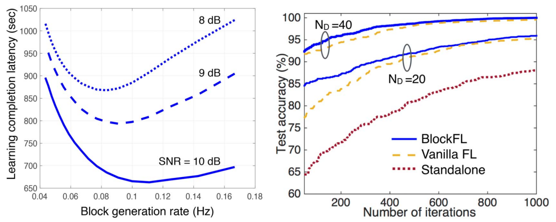
where $N_{\text{fork}}^{(\ell)}$ denotes the number of forking occurrences in the ℓ-th epoch, and follows a geometric distribution with mean $1/(1 - p_{\text{fork}}^{(\ell)})$, with the forking probability $p_{\text{fork}}^{(\ell)}$ at the ℓ-th epoch. Following Step 5, the forking probability is represented as:

$$p_{\text{fork}}^{(\ell)} = 1 - \prod_{M_j \in \mathcal{M}\backslash M_{\hat{o}}} \Pr\left(t_j^{(\ell)} - t_{\hat{o}}^{(\ell)} > t_{\text{bp},j}^{(\ell)} \right), \quad (4)$$

where the term $t_j^{(\ell)} = T_{\text{local},j}^{(\ell)} + T_{\text{up},j}^{(\ell)} + T_{\text{cross},j}^{(\ell)} + T_{\text{bg},j}^{(\ell)}$ is the cumulated delay until the miner M_j generates a block.

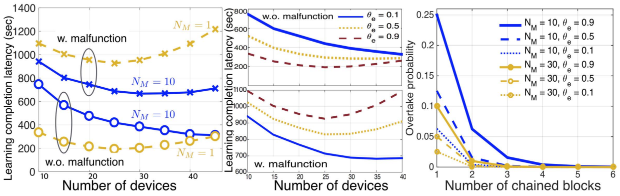
B. Latency Optimal Block Generation Rate

Using the one-epoch latency (3), we derive the optimal block generation rate λ^* that minimizes the ℓ-th epoch latency averaged over the PoW process. Here, the PoW process affects the block generation delay $T_{\text{bg},\hat{o}}^{(\ell)}$, block propagation delay $T_{\text{bp},\hat{o}}^{(\ell)}$, and the number $N_{\text{fork}}^{(\ell)}$ of forking occurrences, which are interdependent due to the mining winner $M_{\hat{o}}$. We consider the case where all miners synchronously start their PoW processes by adjusting T_{wait} such that $T_{\text{cross},o}^{(\ell)} = T_{\text{wait}} - (T_{\text{local},o}^{(\ell)} + T_{\text{up},o}^{(\ell)})$. In this case, even the miners completing the cross-verification earlier wait until T_{wait}, providing the latency upper bound. With this approximation, we derive the optimal block generation rate λ^* as follows.



(a) With respect to λ (b) Test accuracy

Fig. 3. Average learning completion latency (a) versus block generation rate λ and (b) test accuracy of BlockFL, Vanilla FL, and standalone without federation ($\gamma_{\text{up},o} = \gamma_{\text{dn},o} = \gamma_{oj} = \text{SNR}$).



(a) With/without malfunction. (b) For different θ_{e}. (c) Overtake probability.

Fig. 4. Average learning completion latency versus the number of devices, (a) under the miners' malfunction, (b) for different energy constraints θ_e, and (c) overtake probability with respect to the number of chained blocks.

Proposition 1: With the PoW synchronous approximation, i.e., $T^{(\ell)}_{\text{cross},o} = T_{\text{wait}} - (T^{(\ell)}_{\text{local},o} + T^{(\ell)}_{\text{up},o})$, the block generation rate λ^* minimizing the ℓ-th epoch latency $\mathbb{E}[T^{(\ell)}_o]$ averaged over the PoW process is given by:

$$\lambda^* \approx 2\left(T^{(\ell)}_{\text{bp},\hat{o}}\left[1+\sqrt{1+4\ N_M\left(1+T_{\text{ wait}}\,/T^{(\ell)}_{\text{bp},\hat{o}}\right)}\,\right]\right)^{-1}.$$

Proof: Applying the synchronous PoW approximation and the mean $1/(1-p^{(\ell)}_{\text{fork}})$ of the geometrically distributed $N^{(\ell)}_{\text{fork}}$ to (3),

$$\mathsf{E}[T^{(\ell)}_o] \approx \left(T_{\text{wait}} + \mathsf{E}[T^{(\ell)}_{\text{bg},\hat{o}}]\right) / \left(1 - p^{(\ell)}_{\text{fork}}\right) + T^{(\ell)}_{\text{dn},o} + T^{(\ell)}_{\text{global},o}. \tag{5}$$

The terms T_{wait}, $T^{(\ell)}_{\text{dn},o}$, $T^{(\ell)}_{\text{global},o}$ are constant delays given in Sect. II. For the probability $p^{(\ell)}_{\text{fork}}$, using (4) with $t^{(\ell)}_j - t^{(\ell)}_{\hat{o}} = T^{(\ell)}_{\text{bg},j} - T^{(\ell)}_{\text{bg},\hat{o}}$ under the synchronous approximation, we obtain $p^{(\ell)}_{\text{fork}}$ as: $p^{(\ell)}_{\text{fork}} = 1 - e^{\lambda \sum_{M_j \in \mathcal{M} \setminus M_{\hat{o}}} T^{(\ell)}_{\text{bp},j}}$, where $T^{(\ell)}_{\text{bp},j}$ is a constant delay given in Sect. II-A. Next, for the delay $\mathsf{E}[T^{(\ell)}_{\text{bg},\hat{o}}]$, using the definition of $T^{(\ell)}_{\text{bg},\hat{o}}$ and the complementary cumulative distribution function (CCDF) of the exponentially distributed $T^{(\ell)}_{\text{bg},j}$, we derive $T^{(\ell)}_{\text{bg},\hat{o}}$'s CCDF as: $\Pr\left(T^{(\ell)}_{\text{bg},\hat{o}} > x\right) = \prod_{j=1}^{N_M} \Pr\left(T^{(\ell)}_{\text{bg},j} > x\right) = e^{-\lambda N_M x}$. Applying the total probability theorem yields $\mathsf{E}[T^{(\ell)}_{\text{bg},\hat{o}}] = 1/(\lambda N_M)$. Finally, combining all these terms, (5) is recast as: $\mathsf{E}[T^{(\ell)}_o] \approx (T_{\text{wait}} + 1/\lambda N_M) e^{\lambda \sum_{M_j \in \mathcal{M} \setminus M_{\hat{o}}} T^{(\ell)}_{\text{bp},j}} + T^{(\ell)}_{\text{dn},o} + T^{(\ell)}_{\text{global},o}$, which is convex for λ. The optimum λ^* is thus directly derived. ■

For a larger λ, the forking event occurs more frequently, increasing the learning completion latency. On the contrary, for a high PoW difficulty with a low λ, the block generation time incurs its own overheads with excessive latency.

IV. Numerical Results and Discussion

We numerically evaluate the proposed blockFL's average learning completion latency $\mathsf{E}[T_o] = \sum_{\ell=1}^{L} \mathsf{E}[T^{(\ell)}_o]$. By default, we consider $N_D = N_M = 10$, and $N_i \sim \text{Uni}(10, 50)\ \forall D_i \in \mathcal{D}$. Following the 3GPP LTE Cat. M1 specification, we use $W_{\text{up}} = W_{\text{dn}} = W_{\text{m}} = 300$ KHz and $\gamma_{\text{up},o} = \gamma_{\text{dn},o} = \gamma_{oj} = 10$ dB. Other simulation parameters are given as: $\delta_{\text{d}} = 100$ Kbit, $\delta_{\text{m}} = 5$ Kbit, $h = 200$ Kbit, $f_c = 1$ GHz, $T_{\text{wait}} = 50$ ms, $T_{\text{a,wait}} = 500$ ms.

Fig. 3-a shows the impact of block generation rate λ on the BlockFL's average learning completion latency. In Fig. 3-a, we observe that the latency is convex-shaped and is decreasing with the SNRs. For the optimal block generation rate λ^*, the minimized average learning completion latency obtained from Proposition 1 is always longer by up to 1.5% than the simulated minimum latency. In Fig. 3-b, the BlockFL and the vanilla FL achieve almost the same accuracy for an identical N_D. On the other hand, the learning completion latency of our BlockFL is lower than that of the vanilla FL ($N_M = 1$) as in Fig. 4-a, which shows the scalability in terms of the numbers N_M and N_D of miners and devices, respectively. The average learning completion latency is computed for $N_M = 1, 10$ with or without the miners' malfunction. The malfunction is captured by adding Gaussian noise $\mathcal{N}(-0.1, 0.01)$ to each miner's aggregate local model updates with probability 0.05. Without malfunction, a larger N_M increases the latency due to the increase in their cross-verification and block propagation delays. In BlockFL, each miner's malfunction only distorts its associated device's global model update. Such distortion can be restored by federating with other devices that associate with the miners operating normally. Hence, a larger N_M may achieve a shorter latency for $N_M = 10$ with the malfunction.

Fig. 4-a shows that there exists a latency-optimal number N_D of devices. A larger N_D enables to utilize a larger amount of data samples, whereas it increases each block size and block exchange delays, resulting in the convex-shaped latency.

In Fig. 4-b, we assume that some miners cannot participate if their battery level is lower than a predefined threshold value θ_e, a normalized battery level, $\theta_e \in [0,1]$. Without the malfunction of miner nodes, the learning completion latency becomes larger for a lower θ_e due to an increase in cross-verification and block propagation delays. On the contrary, when the malfunction occurs, a lower θ_e achieves a shorter latency because more miners federate with leading to robust global model updates. Fig. 4-c shows the overtake probability that a malicious miner will ever form a new blockchain whose length is longer than a blockchain formed by honest miners. The overtake probability goes to zero if just a few blocks have already been chained by honest miners. Although the malicious miner begins the first PoW with the honest miners, the larger number of miners prevents the overtake.

References

[1] P. Popovski *et al.*, "Wireless access for ultra-reliable low-latency communication: Principles and building blocks," *IEEE Netw.*, vol. 32, no. 2, pp. 16–23, Mar./Apr. 2018.

[2] M. Bennis, M. Debbah, and H. V. Poor, "Ultra-reliable and low-latency wireless communication: Tail, risk and scale," Jan. 2018, *arXiv:1801.01270*. [Online]. Available: https://arxiv.org/abs/1801.01270

[3] J. Park, D. M. Kim, P. Popovski, and S.-L. Kim, "Revisiting frequency reuse towards supporting ultra-reliable ubiquitous-rate communication," in *Proc. 5th Int. Symp. Modeling Optim. Mobile, Ad Hoc, Wireless Netw. (WiOpt)*, May 2017, pp. 1–6.

[4] J. Konečný, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," Oct. 2016, *arXiv:1610.02527*. [Online]. Available: https://arxiv.org/abs/1610.02527

[5] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. AISTATS*, Fort Lauderdale, FL, USA, Apr. 2017, pp. 1–11.

[6] J. Park, S. Samarakoon, M. Bennis, and M. Debbah, "Wireless network intelligence at the edge," *Proc. IEEE*, to be published. [Online]. Available: https://arxiv.org/abs/1812.02858

[7] H. B. McMahan and D. Ramage. (2017). *Federated Learning: Collaborative Machine Learning Without Centralized Training Data*. Accessed: Oct. 2008. [Online]. Available: https://ai.googleblog.com/2017/04/federated-learning-collaborative.html

[8] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[9] C. Decker and R. Wattenhofer, "Information propagation in the Bitcoin network," in *Proc. IEEE P2P*, Trento, Italy, Sep. 2013, pp. 1–10.

[10] S. Samarakoon, M. Bennis, W. Saad, and M. Debbah, "Distributed federated learning for ultra-reliable low-latency vehicular communications," Jul. 2018, *arXiv:1807.08127*. [Online]. Available: https://arxiv.org/abs/1807.08127

[11] P. Danzi, A. E. Kalor, C. Stefanovic, and P. Popovski, "Analysis of the communication traffic for blockchain synchronization of IoT devices," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 1–7.

[12] N. C. Luong, D. Niyato, P. Wang, and Z. Xiong, "Optimal auction for edge computing resource management in mobile blockchain networks: A deep learning approach," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 1–6.

[13] L. Chen, L. Xu, N. Shah, Z. Gao, Y. Lu, and W. Shi, "On security analysis of proof-of-elapsed-time," in *Proc. Int. Symp. Stabilization, Saf., Secur. Distrib. Syst.*, Boston, MA, USA, Nov. 2017, pp. 282–297.