# TiBroco: A Fast and Secure Distributed Learning Framework for Tiered Wireless Edge Networks

Dong-Jun Han, Jy-yong Sohn, and Jaekyun Moon
School of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST)
Email: {djhan93, jysohn1108}@kaist.ac.kr, jmoon@kaist.edu

*Abstract*—**Recent proliferation of mobile devices and edge servers (e.g., small base stations) strongly motivates distributed learning at the wireless edge. In this paper, we propose a fast and secure distributed learning framework that utilizes computing resources at edge servers as well as distributed computing devices in tiered wireless edge networks. A fundamental lower bound is derived on the computational load that perfectly tolerates Byzantine attacks at both tiers. TiBroco, a hierarchical coding framework achieving this theoretically minimum computational load is proposed, which guarantees *secure* distributed learning by combating Byzantines. A *fast* distributed learning is possible by precisely allocating loads to the computing devices and edge servers, and also utilizing the broadcast nature of wireless devices. Extensive experimental results on Amazon EC2 indicate that our TiBroco allows significantly faster distributed learning than existing methods while guaranteeing full tolerance against Byzantine attacks at both tiers.**

## I. Introduction

Distributed learning [1]–[4] has now become a key player for reducing the training time of large-scale machine learning applications such as computer vision, speech recognition and recommendation systems. Due to the advent of mobile edge computing (MEC), the trend on distributed computing/learning is recently moving from current cloud-based systems toward edge-facilitated systems [5]–[7]; small base stations (BSs) or access points integrated with an MEC server are valuable computational resources for distributed learning, as they are capable of handling a large volume of data. Moreover, not only small BSs but also billions of mobile and Internet of Things (IoT) devices of the wireless networks are potential resources for distributed learning.

However, Byzantine attacks [8], [9] are one of the main bottlenecks in edge-facilitated distributed learning systems. During computation of local gradients, distributed nodes affected by Byzantine attacks may release arbitrary and faulty outputs rather than true gradient values, severely degrading the performance of learning. These Byzantine attacks are frequently observed in the wireless network where a number of low-cost IoT devices coexist [10], [11]. It is also reported that even BSs such as femtocell BSs are vulnerable to Byzantines [12], [13].

To alleviate the effect of Byzantine attacks, various median-based aggregation methods have been proposed in the literature in a simple parameter server (PS) framework [14]–[19]. While these approaches show performance improvements compared to naive averaging, some strong assumptions are required to guarantee convergence. Moreover, as the number of computing nodes increases, the median-based methods require significant computational cost at the PS. Based on these observations, a coding-theoretic approach was proposed in [20] which employs redundant computation at the nodes. This coding-based method theoretically guarantees perfect tolerance against Byzantines with significantly smaller training time compared to the median-based aggregation methods.

In this paper, we also tackle the Byzantine failure problem but in a practical wireless setup where gradient computation is performed at the individual devices, the end nodes of the network, as well as at edge servers (like small BSs and access points). We develop TiBroco, a coded distributed learning framework specifically geared to the tiered and broadcast nature of wireless edge networks, to allow fast and secure learning despite severe Byzantine attacks at both tiers. Compared to a simple PS framework assumed in [20], our model explicitly reflects the hierarchical architecture of current edge computing systems [21], [22]. Each edge server (ES) or BS forms a cell, and computing nodes such as mobile/IoT devices are connected wirelessly to these ESs. The ESs are connected to a PS, which can be viewed as a part of fog or cloud. Here, both devices and ESs can be affected by Byzantine attacks[1]. Another characteristic of the system we consider is the existence of wireless devices in the overlapping areas between adjacent cells [23], which arises from dense deployment of ESs in 5G and beyond systems. A number of mobile/IoT devices located in this overlapping region can send the computed results to more than one ES through a wireless broadcast channel. Under this hierarchical setup with broadcasting devices in overlapping zones, we aim at speeding up distributed learning while mitigating the effect of Byzantine attacks on computing devices and ESs in the system.

**Main contributions.** We propose TiBroco, an edge-facilitated distributed learning framework which utilizes 1) two-tier computing (leveraging computing powers of both ESs and devices), 2) hierarchical coding (allowing redundant computation at the nodes to combat Byzantine attacks at both

This work was supported by National Research Foundation of Korea (No. 2019R1I1A2A02061135).

[1]We assume that in a given learning session on a particular dataset, all participating ESs and computing devices themselves are authenticated and trustworthy, but Byzantine attacks may affect the transmitted messages of the devices or the ESs (e.g., by corrupting the corresponding allocated frequency or time slots in cellular networks). In this context, Byzantine devices and ESs in this paper refer to those whose transmissions are compromised by arbitrary Byzantine attacks.

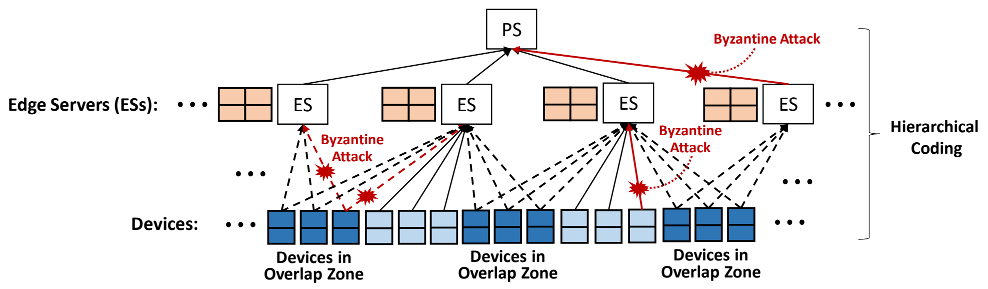


Fig. 1. TiBroco: the proposed distributed learning framework leveraging two-tier computing of edge servers and devices, hierarchical coding and broadcast nature of wireless devices in overlapping zones. The gradients for individual data partitions (represented by the boxes) are computed at edge servers and devices and are collected at the PS. The goal is to provide a protection against Byzantine attacks at both edge servers and devices, via redundant data allocation and gradient computations. By increasing the load at the ESs, one can reduce the burden at the devices while combating the same number of Byzantines. The devices in the overlapping areas transmit their results to both ESs by broadcasting.

tiers) and 3) broadcast nature of wireless devices (located in the overlapping cell region). The goal is to combat any $b_1$ Byzantine devices and any $b_2$ Byzantine ESs in the system (the goal can also be setup in terms of the probability of a node being Byzantine). A high-level description of TiBroco is given in Fig. 1. The PS which has the whole dataset allocates data to the distributed nodes to exploit the distributed computing resources. Each data partition is allocated to multiple ESs/devices, inducing redundant gradient computation. The computed gradients at individual device are linearly combined according to a certain rule and sent to its ES. Here, some devices are in the overlapping zone and can broadcast to more than one ES. Each ES linearly combines the received gradients along with the ones it computes on its own, and sends the result to the PS. Byzantine attacks arise at both devices and ESs, and coding (i.e., redundant data allocation) must be carefully designed to provide protection at both tiers. Coding must also exploit the wireless nature of the network, where some devices can broadcast to more than one ES, allowing a reduced redundancy for a given level of protection.

Our approach is to first derive the theoretical bound on computational load to combat any $b_1$ Byzantine devices and any $b_2$ Byzantine ESs in the system. We explore trade-off between computational loads at devices and ESs to combat Byzantines; as more load is allocated to the ES, one can reduce the burden at the device while combating the same number of Byzantines. Then we propose a hierarchical coding scheme achieving this bound, which guarantees *secure* distributed learning.

A *fast* distributed learning is possible by broadcasting and appropriate load allocation. We utilize the broadcast nature of wireless networks to speed up training by reducing the communication rounds at the devices in the overlapping cell regions. Finally, we provide optimal load allocation solutions at the devices and the ESs to reduce the training time further.

We implement TiBroco on Amazon EC2 for training ResNet18 and a logistic regression model. Extensive experimental results show that TiBroco is much faster than the median-based approach and existing coding idea while guaranteeing perfect tolerance against Byzantine attacks.

**Related works.** Coding theory is regarded as a promising tool for overcoming bottlenecks of distributed computing/learning via redundant computation. These redundancy-based approaches have been extensively studied in recent years targeting straggler issues [23]–[30] and Byzantine issues [20], [31]–[33]. A closely related scheme to our work is DRACO [20], which provides a coding scheme to combat Byzantines. Compared to the existing median-based approaches, DRACO guarantees perfect tolerance against any Byzantine attacks and requires less complexity at the PS. DETOX [33] is another work which utilizes the idea of redundant computation along with the median-based aggregation. However, the design of these schemes does not reflect the practical structure of the wireless edge network; these schemes do not fare well in current edge-facilitated learning systems. Providing new design strategies leveraging computing resources of both ESs and devices (two-tier edge computing), redundancies at the nodes to combat Byzantines both tiers (hierarchical coding), and the existence of broadcasting devices in overlapping cell areas (wireless broadcast) are the unique contributions of our work.

Finally, we note that our recent work [23] also considers a wireless setup but tackles the straggler issue. In the current paper, we tackle the Byzantine attack issue which makes the problem more difficult compared to the setup with stragglers.

**Notations.** Let $\mathbf{1}_{a\times b}$ denote an $a\times b$ all-ones matrix and $\mathbf{0}_{a\times b}$ as an $a\times b$ all-zeros matrix. We also define $[n] := \{1, 2, \dots, n\}$.

## II. PROPOSED CODED DISTRIBUTED LEARNING SCHEME

### A. Background: Distributed Learning

Given a dataset $D$ with $m$ sample points and a loss function $\ell(\cdot)$, we would like to solve $\min_{\mathbf{w}\in\mathbb{R}^d} \sum_{i=1}^m \ell(\mathbf{w}; \mathbf{x}_i)$, where $\mathbf{x}_i$ is the $i$-th data point. We iteratively update the model as

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \gamma \mathbf{g}^{(t)}, \quad (1)$$

where $\mathbf{g}^{(t)} := \sum_{i=1}^m \nabla\ell(\mathbf{w}^{(t)}; \mathbf{x}_i)$ and $\gamma$ is the learning rate. Let $k$ be the number of data partitions, i.e., $D = \{D_i\}_{i=1}^k$ and $n$ be the number of computing nodes in the system. Then, $\frac{k}{n}$ data partitions are assigned to each node, in the case of no redundancy, where $k = n$ is generally assumed. Each node computes the sum of gradients of the assigned data partitions and transmits to the PS. After aggregating the results from all computing nodes, the PS clearly obtains $\mathbf{g}^{(t)} = \sum_{i=1}^k \mathbf{g}_i^{(t)}$, where $\mathbf{g}_i^{(t)} := \sum_{\mathbf{x}\in D_i} \nabla\ell(\mathbf{w}^{(t)}; \mathbf{x})$ is the sum of gradients of data points in $D_i$. Now the PS updates the model according to (1), distributes the updated model $\mathbf{w}^{(t+1)}$ to the computing nodes, and then move on to the next iteration. For the rest of the paper, we will drop index $t$ for notational simplicity. Our goal is to obtain the full gradient $\sum_{i=1}^{k}\mathbf{g}_i$ at the PS.

## B. TiBroco with Edge Computing, Hierarchical Coding and Broadcasting Devices

In this paper, we consider a wireless setup where the PS is connected to $L$ ESs, each of which serves computing devices in its covered area or *cell*. Due to dense deployment of ESs, there typically exist overlapping areas between adjacent cells. The computing devices in the overlapping cell areas can be connected to more than one ES. Let $u_i$ be the number of computing devices connected only to the $i$-th ES. For $i \neq j$, define $v_{i,j}$ (or $v_{j,i}$) as the number of devices connected to both ES $i$ and ES $j$, located in the overlapping area between two cells. Each device is assumed to be connected to at most two ESs, although the proposed coding idea can be easily extended to general cell topologies with more overlapping ESs following exactly the same procedure. Now the number of devices in cell $i$, denoted as $f_i$, is written as $f_i = u_i + \sum_{j\in[L]\setminus\{i\}} v_{i,j}$. The total number of devices in the system becomes

$$n = \sum_{i=1}^{L} f_i - \sum_{i=1}^{L}\sum_{j\in[L],\ j<i} v_{i,j}. \quad (2)$$

Fig. 2 shows the high-level description of TiBroco with $L = 4$, $u_i = 5$ for all $i \in [L]$ and $v_{1,2} = v_{2,3} = v_{3,4} = v_{4,1} = 2$, $v_{1,3} = v_{2,4} = 0$. Based on some predefined coding rule, the overall dataset is distributed across $n$ devices and $L$ ESs in the system to exploit the distributed computing resources. The goal is to obtain the exact sum $\sum_{i=1}^{k}\mathbf{g}_i$ at the PS[2]. We have the following definition on computational loads.

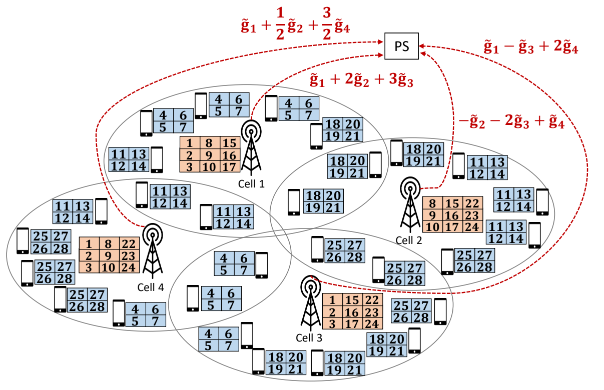
**Definition 1.** *$r_s$ and $r_w$ are the number of data partitions computed at each ES (seen as a submaster) and each device (seen as a worker), respectively.*

The examples of Figs. 2(a) and 2(b) illustrate the cases of $(r_s, r_w) = (9, 4)$ and $(r_s, r_w) = (15, 2)$.

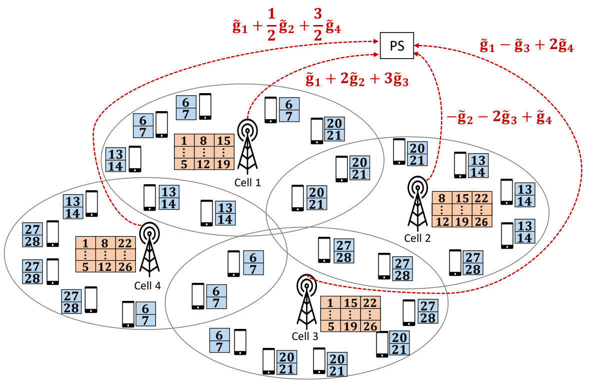
Now we describe TiBroco, for tolerating any $b_1$ Byzantine devices and any $b_2$ Byzantine ESs in the system. Let $\mathbf{G} = [\mathbf{g}_1, \mathbf{g}_2, \dots, \mathbf{g}_k]^T$ be a $k \times d$ matrix having the $d$-dimensional gradient of each data partition in its row. PS first designs an $L \times k$ encoding matrix $\mathbf{A}$, which allocates $k$ data partitions to $L$ cells. Here, matrix $\mathbf{A}$ should be designed so that $\sum_{i=1}^{k}\mathbf{g}_i$ can be recovered in the presence of $b_2$ Byzantine ESs. At the same time, the design should also reflect the data partitions shared by adjacent cells. To begin, the $i$-th row of $\mathbf{AG}$ should correspond to the desired result that the PS gets cell $i$. For the example of Fig. 2, we have

$$\mathbf{AG} = \begin{bmatrix}\mathbf{a}_1\mathbf{G}\\ \mathbf{a}_2\mathbf{G}\\ \mathbf{a}_3\mathbf{G}\\ \mathbf{a}_4\mathbf{G}\end{bmatrix} = \begin{bmatrix}\tilde{\mathbf{g}}_1^T + 2\tilde{\mathbf{g}}_2^T + 3\tilde{\mathbf{g}}_3^T\\ -\tilde{\mathbf{g}}_2^T - 2\tilde{\mathbf{g}}_3^T + \tilde{\mathbf{g}}_4^T\\ \tilde{\mathbf{g}}_1^T - \tilde{\mathbf{g}}_3^T + 2\tilde{\mathbf{g}}_4^T\\ \tilde{\mathbf{g}}_1^T + \frac{1}{2}\tilde{\mathbf{g}}_2^T + \frac{3}{2}\tilde{\mathbf{g}}_4^T\end{bmatrix}.$$

[2]TiBroco is also applicable to mini-batch stochastic gradient descent (SGD), by viewing $k$ as the number of data samples/partitions in each mini-batch.


$\tilde{\mathbf{g}}_1 + \frac{1}{2}\tilde{\mathbf{g}}_2 + \frac{3}{2}\tilde{\mathbf{g}}_4$
PS
$\tilde{\mathbf{g}}_1 - \tilde{\mathbf{g}}_3 + 2\tilde{\mathbf{g}}_4$
$\tilde{\mathbf{g}}_1 + 2\tilde{\mathbf{g}}_2 + 3\tilde{\mathbf{g}}_3$
$-\tilde{\mathbf{g}}_2 - 2\tilde{\mathbf{g}}_3 + \tilde{\mathbf{g}}_4$
Cell 1
Cell 2
Cell 3
Cell 4

(a) $r_s = 9$, $r_w = 4$


$\tilde{\mathbf{g}}_1 + \frac{1}{2}\tilde{\mathbf{g}}_2 + \frac{3}{2}\tilde{\mathbf{g}}_4$
PS
$\tilde{\mathbf{g}}_1 - \tilde{\mathbf{g}}_3 + 2\tilde{\mathbf{g}}_4$
$\tilde{\mathbf{g}}_1 + 2\tilde{\mathbf{g}}_2 + 3\tilde{\mathbf{g}}_3$
$-\tilde{\mathbf{g}}_2 - 2\tilde{\mathbf{g}}_3 + \tilde{\mathbf{g}}_4$
Cell 1
Cell 2
Cell 3
Cell 4

(b) $r_s = 15$, $r_w = 2$

Fig. 2. TiBroco framework leveraging two-tier coding and wireless edge computing, with $n = k = 28$, $L = 4$, $b_1 = 1$, $b_2 = 1$. The square with index $i$ corresponds to data partition $D_i$. The partial sums are defined as $\tilde{\mathbf{g}}_1 = \sum_{i=1}^{7}\mathbf{g}_i$, $\tilde{\mathbf{g}}_2 = \sum_{i=8}^{14}\mathbf{g}_i$, $\tilde{\mathbf{g}}_3 = \sum_{i=15}^{21}\mathbf{g}_i$, $\tilde{\mathbf{g}}_4 = \sum_{i=22}^{28}\mathbf{g}_i$. By increasing the load at the ESs, one can reduce the burden at the devices while combating the same number of Byzantines. The devices in the overlapping areas transmit their results to both ESs by broadcasting.

ES $i$ sends $\mathbf{a}_i\mathbf{G}$ to the PS, but if this ES happens to be Byzantine, then it would instead send $\mathbf{a}_i\mathbf{G} + \mathbf{n}_i$, where $\mathbf{n}_i$ is some $1 \times d$ vector that is not all-zero.

Based on the encoding matrix $\mathbf{A}$, we also need to define a decoder function $S$ for obtaining $\sum_{i=1}^{k}\mathbf{g}_i$ at the PS based on the collected results from $L$ ESs, in the possible presence of $b_2$ Byzantine ESs. Hence, the goal at the PS is to design $\mathbf{A}$ and $S(\cdot)$ such that

$$S(\mathbf{AG} + \mathbf{N}) = \sum_{i=1}^{k}\mathbf{g}_i^T, \quad (3)$$

where $\mathbf{N} = [\mathbf{n}_1, \mathbf{n}_2, \dots, \mathbf{n}_L]^T$. With a maximum of $b_2$ Byzantine ESs in the system, we have $|\{i : ||\mathbf{n}_i|| \neq 0\}| \leq b_2$. In the example in Fig. 2, it is easy to verify that $\sum_{i=1}^{k}\mathbf{g}_i$ can be obtained by linearly combining any two clean vectors of $\mathbf{a}_1\mathbf{G}$, $\mathbf{a}_2\mathbf{G}$, $\mathbf{a}_3\mathbf{G}$, and $\mathbf{a}_4\mathbf{G}$. Hence, by defining the decoder function $S$ to be majority voting on all six pair-wise sums, the system is able to tolerate any $b_2 = 1$ Byzantine ES.

Now consider a specific cell $i$. The goal of ES $i$ is to recover $\mathbf{a}_i\mathbf{G}$ against $b_1$ Byzantine devices. We define an $f_i \times k$ encoding matrix $\mathbf{B}^{(i)}$ which allocates data partitions to $f_i$ end device nodes in cell $i$. Here, the $j$-th row of $\mathbf{B}^{(i)}\mathbf{G}$, labeled $\mathbf{b}_j^{(i)}\mathbf{G}$, is the desired result that ES $i$ gets from the $j$-th node in cell $i$. For example in Fig. 2(a), we have

$$\mathbf{B}^{(1)}\mathbf{G} = \begin{bmatrix} \mathbf{b}_1^{(1)}\mathbf{G} \\ \mathbf{b}_2^{(1)}\mathbf{G} \\ \mathbf{b}_3^{(1)}\mathbf{G} \\ \mathbf{b}_4^{(1)}\mathbf{G} \\ \mathbf{b}_5^{(1)}\mathbf{G} \\ \mathbf{b}_6^{(1)}\mathbf{G} \\ \mathbf{b}_7^{(1)}\mathbf{G} \\ \mathbf{b}_8^{(1)}\mathbf{G} \\ \mathbf{b}_9^{(1)}\mathbf{G} \end{bmatrix} = \begin{bmatrix} \mathbf{g}_4^T + \mathbf{g}_5^T + \mathbf{g}_6^T + \mathbf{g}_7^T \\ \mathbf{g}_4^T + \mathbf{g}_5^T + \mathbf{g}_6^T + \mathbf{g}_7^T \\ \mathbf{g}_4^T + \mathbf{g}_5^T + \mathbf{g}_6^T + \mathbf{g}_7^T \\ \mathbf{g}_{11}^T + \mathbf{g}_{12}^T + \mathbf{g}_{13}^T + \mathbf{g}_{14}^T \\ \mathbf{g}_{11}^T + \mathbf{g}_{12}^T + \mathbf{g}_{13}^T + \mathbf{g}_{14}^T \\ \mathbf{g}_{11}^T + \mathbf{g}_{12}^T + \mathbf{g}_{13}^T + \mathbf{g}_{14}^T \\ \mathbf{g}_{18}^T + \mathbf{g}_{19}^T + \mathbf{g}_{20}^T + \mathbf{g}_{21}^T \\ \mathbf{g}_{18}^T + \mathbf{g}_{19}^T + \mathbf{g}_{20}^T + \mathbf{g}_{21}^T \\ \mathbf{g}_{18}^T + \mathbf{g}_{19}^T + \mathbf{g}_{20}^T + \mathbf{g}_{21}^T \end{bmatrix}$$

from cell 1. Here, by Definition 1, each row of $\mathbf{B}^{(i)}$ should have $r_w$ non-zero values. Considering the effect of Byzantine devices, the received signal at ES $i$ becomes $\mathbf{B}^{(i)}\mathbf{G} + \mathbf{M}^{(i)}$, where $\mathbf{M}^{(i)} = [\mathbf{m}_1^{(i)}, \mathbf{m}_2^{(i)}, \dots, \mathbf{m}_{f_i}^{(i)}]^T$ and $\mathbf{m}_j^{(i)}$ can be an arbitrary $1 \times d$ vector if node $j$ in cell $i$ is Byzantine and $\mathbf{m}_j^{(i)} = \mathbf{0}_{1\times d}$, otherwise. Assuming a maximum of $b_1$ Byzantine devices in each cell, $|\{j : ||\mathbf{m}_j^{(i)}|| \neq 0\}| \leq b_1$ holds.

Note that the devices in the overlapped region between cell $i$ and cell $j$ receive data partitions from both ESs. We design $\mathbf{B}^{(i)}$ and $\mathbf{B}^{(j)}$ such that each device in the overlapped area receives the same encoding coefficients from both ESs. The device can simply *broadcast* the computed gradients to both ESs with one communication round.

While the devices are computing gradients, each ES also performs computation for the assigned data partitions. We define an $1 \times k$ data allocation vector $\mathbf{c}^{(i)}$, which allocates data partitions to ES $i$. ES $i$ computes gradients to obtain $\mathbf{c}^{(i)}\mathbf{G}$. For example in Fig. 2(a), we have

$$\begin{aligned}\mathbf{c}^{(1)}\mathbf{G} &= \mathbf{g}_1^T + \mathbf{g}_2^T + \mathbf{g}_3^T + \mathbf{g}_4^T + \mathbf{g}_5^T \\ &\quad + 2(\mathbf{g}_8^T + \mathbf{g}_9^T + \mathbf{g}_{10}^T + \mathbf{g}_{11}^T + \mathbf{g}_{12}^T) \\ &\quad + 3(\mathbf{g}_{15}^T + \mathbf{g}_{16}^T + \mathbf{g}_{17}^T + \mathbf{g}_{18}^T + \mathbf{g}_{19}^T).\end{aligned}$$

Again by Definition 1, the number of non-zero values of $\mathbf{c}^{(i)}$ should be $r_s$. Based on $\mathbf{B}^{(i)}$ and $\mathbf{c}^{(i)}$, we design a decoder function at $Q_i$ to recover $\mathbf{a}_i G$ from the ES's own computation as well as the collected results from devices in its covered area. Hence, the goal at ES $i$ is to design $\mathbf{B}^{(i)}$, $\mathbf{c}^{(i)}$, $Q_i(\cdot)$ such that

$$\mathbf{a}_i\mathbf{G} = Q_i(\mathbf{B}^{(i)}\mathbf{G} + \mathbf{M}^{(i)}) + \mathbf{c}^{(i)}\mathbf{G} \tag{4}$$

where $|\{j : ||\mathbf{m}_j^{(i)}|| \neq 0\}| \leq b_1$. Note that the devices in each cell can be divided into three groups having the same data partitions in Fig. 2. By defining the decoder function $Q_i$ to be majority voting on each group in cell $i$, each cell is able to tolerate any $b_1 = 1$ Byzantine devices. By taking the weighted sum of decoded result and $\mathbf{c}^{(i)}\mathbf{G}$, each ES can successfully recover the desired vector $\mathbf{a}_i\mathbf{G}$.

One important observation from Fig. 2 is the trade-off between computational burden at the devices and ESs. Both systems in Figs. 2(a) and 2(b) can combat the same numbers of Byzantines at two tiers with different computational loads. By properly offloading loads to the ESs, we can reduce the burden at the devices while tolerating the same numbers of Byzantines at two tiers.

To summarize, the goal of this paper is to design the coding or data allocation rules $\mathbf{A}$, $\{\mathbf{B}^{(i)}\}$ and $\{\mathbf{c}^{(i)}\}$ as well as the matching decoding functions $S$ and $\{Q_i\}$ such that (3) and (4) hold, i.e., any $b_1$ Byzantine computing devices and any $b_2$ ESs are tolerated in the system with a minimum computational burden (along with appropriate load balancing of $r_s$ and $r_w$).

## III. Theoretical Guarantee and Practical Code Construction

### A. Fundamental Lower Bound

We first describe our main theorem, which provides a fundamental lower bound on computational load to combat Byzantines.

**Theorem 1.** *Suppose a scheme is able to obtain the exact sum of gradients $\sum_{i=1}^k \mathbf{g}_i$ at the PS against any $b_1$ Byzantine computing devices and $b_2$ Byzantine ESs in the system. Then, we must have*

$$r_w \geq \frac{(2b_1+1)\{k(2b_2+1) - r_s L\}}{\sum_{i=1}^L f_i} := r_w^*, \tag{5}$$

*where $f_i$ is the number of devices in cell $i$.*

*Proof:* To combat $b_2$ Byzantine ESs in the system, each data partition $D_j \in D = \{D_1, D_2, ...D_k\}$ should be replicated to at least $2b_2+1$ cells [20]. The intuition is as follows. Assume that data partition $D_j$ is replicated to $c$ cells, where $c \leq 2b_2$. If $c \leq b_2$, the scheme is clearly not robust to $b_2$ Byzantine ESs when all ESs having $D_j$ are Byzantines. Now consider the case $b_2 \leq c \leq 2b_2$. Since we have $\frac{1}{2}c \leq b_2$, if $b_2$ ESs containing $D_j$ are Byzantines, which are the majority, the scheme is not able to combat $b_2$ Byzantine ESs. By defining $\mu_i$ as the number of data partitions allocated to the $i$-th cell, we have

$$\sum_{i=1}^L \mu_i \geq k(2b_2+1). \tag{6}$$

Now focus on a specific $i$-th cell, where $r_s$ out of $\mu_i$ data partitions are computed by the corresponding ES. The remaining $\mu_i - r_s$ data partitions should be allocated to the computing devices in cell $i$, where each should be replicated at least $2b_1+1$ times to combat $b_1$ Byzantine computing devices in each cell. By defining $r_w$ as the number of data partitions allocated to each computing node, we have $f_i r_w \geq (\mu_i - r_s)(2b_1+1)$ for all $i \in [L]$. By summing up for all cell indices $i \in [L]$, we can write

$$r_w \sum_{i=1}^L f_i \geq \sum_{i=1}^L (\mu_i - r_s)(2b_1+1) \tag{7}$$

$$= (2b_1+1)\{\sum_{i=1}^L \mu_i - Lr_s\} \tag{8}$$

$$\geq (2b_1+1)\{k(2b_2+1) - Lr_s\} \tag{9}$$

where (9) comes from (6). Dividing both sides by $\sum_{i=1}^{L} f_i$, we have $r_w \geq \frac{(2b_1+1)\{k(2b_2+1)-r_s L\}}{\sum_{i=1}^{L} f_i}$, which completes the proof. ■

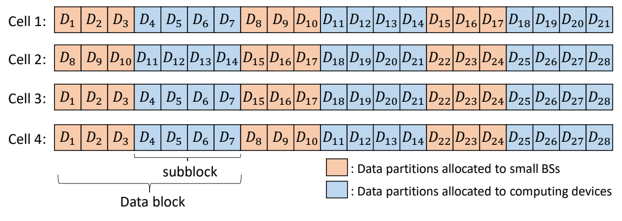
It can be seen from (5) that there exists a trade-off between $r_s$ and the lower bound on $r_w$, i.e., $r_w^*$. If the ESs perform more gradient computations (i.e., increase $r_s$), the lower bound on computational load at the devices is reduced while combating the same number of Byzantines, and vice versa. Note that this bound decreases as more devices exist in the overlapped regions, since for a given $n$ the $\sum_{i=1}^{L} f_i$ increases with more devices in the overlapped zone.

The result in Theorem 1 is fundamental, which means that if the computing devices compute less than $r_w^*$ data partitions, it is impossible to design a scheme $\left(\mathbf{A}, S, \{\mathbf{B}^{(i)}\}, \{\mathbf{c}^{(i)}\}, \{Q_i\}\right)$ that can perfectly tolerate Byzantines. This bound generalizes Theorem 3.1 in [20] for our wireless setup; the special case with $f_i = \frac{n}{L}$ (no overlapping regions), $b_2 = 0$ (no hierarchy), $r_s = 0$ (no computing at ES) reduces to the problem considered in [20].

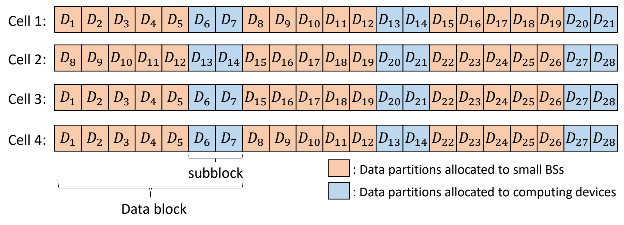
## *B. Coding Scheme Achieving the Fundamental Bound*

In this subsection, we provide a hierarchical coding scheme achieving the theoretical bound, which strategically allocates data partitions to the ESs and the individual devices.

**Step 1. Design of $\mathbf{A}$ and $S$:** The first step is to design matrix $\mathbf{A} \in \mathbb{R}^{L\times k}$ which allocates $k$ data partitions from the PS to $L$ cells. The goal is to obtain $\sum_{i=1}^{k} \mathbf{g}_i$ at the PS in the presence of $b_2$ Byzantine ESs. Matrix $\mathbf{A}$ should be also precisely designed to allow shared data partitions between adjacent cells. For code construction, we let each ES to select the same number of devices for participation, i.e., $f_i = f$ for all $i \in [L]$. We choose $\mathbf{A}$ to have a cyclic structure, where each row of $\mathbf{A}$ has $\frac{k(2b_2+1)}{L}$ consecutive non-zero elements as in the cyclic code proposed in [20]. The consecutive non-zero elements of the $i$-th row are cyclically shifted $\frac{k}{L}$ steps to the right to construct the $(i+1)$-th row. This would mean that there are $\frac{2kb_2}{L}$ data partitions shared by two adjacent cells, some of which will be allocated to the devices in the overlapping region. The exact values of these non-zero elements are determined by utilizing an inverse discrete Fourier transformation matrix as in [20]. We define the first $L-2b_2$ rows of $\mathbf{F}$ as $\mathbf{F}_x$, and the remaining $2b_2$ rows as $\mathbf{F}_y$. We first construct matrix $\mathbf{A}' \in \mathbb{R}^{L\times L}$, a cyclically structured matrix with $2b_2+1$ non-zero elements in each row. The $i$-th row of $\mathbf{A}'$ is shifted one step to the right to yield the $(i+1)$-th row. We define $\alpha_i$ as the set of row indices having zero in the $i$-th column of matrix $\mathbf{A}'$, where $|\alpha_i| = L-2b_2-1$. Since any $L-2b_2$ columns of $\mathbf{F}_x$ are linearly independent, there exists a unique solution $\mathbf{e}_i \in \mathbb{R}^{1\times(L-2b_2-1)}$ satisfying $[\mathbf{e}_i \;\; 1]\cdot[\mathbf{F}_x]_{\cdot,\alpha_i} = \mathbf{0}_{1\times(L-2b_2-1)}$. Now by defining an $L\times(L-2b_2-1)$ matrix $\mathbf{E} = [\mathbf{e}_1; \;\; \mathbf{e}_2; \;\; \cdots \;\; ; \mathbf{e}_L]$, we construct $\mathbf{A}'$ as $\mathbf{A}' = [\mathbf{E} \;\; \mathbf{1}_{L\times 1}]\cdot \mathbf{F}_x$. This guarantees the span of any $L-2b_2$ columns of $\mathbf{A}'$ to contain $\mathbf{1}_{L\times 1}$. It also guarantees $\mathbf{A}'$ to have a cyclic structure. By replicating each element of $(\mathbf{A}')^T$ by $\frac{k}{L}$ times row-wise, we finally obtain an $L\times k$ matrix $\mathbf{A}$ having a cyclic structure with the span of any $L-2b_2$ rows containing $\mathbf{1}_{1\times k}$.


Cell 1: $D_1$ $D_2$ $D_3$ $D_4$ $D_5$ $D_6$ $D_7$ $D_8$ $D_9$ $D_{10}$ $D_{11}$ $D_{12}$ $D_{13}$ $D_{14}$ $D_{15}$ $D_{16}$ $D_{17}$ $D_{18}$ $D_{19}$ $D_{20}$ $D_{21}$
Cell 2: $D_8$ $D_9$ $D_{10}$ $D_{11}$ $D_{12}$ $D_{13}$ $D_{14}$ $D_{15}$ $D_{16}$ $D_{17}$ $D_{18}$ $D_{19}$ $D_{20}$ $D_{21}$ $D_{22}$ $D_{23}$ $D_{24}$ $D_{25}$ $D_{26}$ $D_{27}$ $D_{28}$
Cell 3: $D_1$ $D_2$ $D_3$ $D_4$ $D_5$ $D_6$ $D_7$ $D_{15}$ $D_{16}$ $D_{17}$ $D_{18}$ $D_{19}$ $D_{20}$ $D_{21}$ $D_{22}$ $D_{23}$ $D_{24}$ $D_{25}$ $D_{26}$ $D_{27}$ $D_{28}$
Cell 4: $D_1$ $D_2$ $D_3$ $D_4$ $D_5$ $D_6$ $D_7$ $D_8$ $D_9$ $D_{10}$ $D_{11}$ $D_{12}$ $D_{13}$ $D_{14}$ $D_{22}$ $D_{23}$ $D_{24}$ $D_{25}$ $D_{26}$ $D_{27}$ $D_{28}$
subblock
Data block
: Data partitions allocated to small BSs
: Data partitions allocated to computing devices

(a) $r_s = 9$, $r_w = 4$


Cell 1: $D_1$ $D_2$ $D_3$ $D_4$ $D_5$ $D_6$ $D_7$ $D_8$ $D_9$ $D_{10}$ $D_{11}$ $D_{12}$ $D_{13}$ $D_{14}$ $D_{15}$ $D_{16}$ $D_{17}$ $D_{18}$ $D_{19}$ $D_{20}$ $D_{21}$
Cell 2: $D_8$ $D_9$ $D_{10}$ $D_{11}$ $D_{12}$ $D_{13}$ $D_{14}$ $D_{15}$ $D_{16}$ $D_{17}$ $D_{18}$ $D_{19}$ $D_{20}$ $D_{21}$ $D_{22}$ $D_{23}$ $D_{24}$ $D_{25}$ $D_{26}$ $D_{27}$ $D_{28}$
Cell 3: $D_1$ $D_2$ $D_3$ $D_4$ $D_5$ $D_6$ $D_7$ $D_{15}$ $D_{16}$ $D_{17}$ $D_{18}$ $D_{19}$ $D_{20}$ $D_{21}$ $D_{22}$ $D_{23}$ $D_{24}$ $D_{25}$ $D_{26}$ $D_{27}$ $D_{28}$
Cell 4: $D_1$ $D_2$ $D_3$ $D_4$ $D_5$ $D_6$ $D_7$ $D_8$ $D_9$ $D_{10}$ $D_{11}$ $D_{12}$ $D_{13}$ $D_{14}$ $D_{22}$ $D_{23}$ $D_{24}$ $D_{25}$ $D_{26}$ $D_{27}$ $D_{28}$
subblock
Data block
: Data partitions allocated to small BSs
: Data partitions allocated to computing devices

(b) $r_s = 15$, $r_w = 2$

Fig. 3. Data allocation with different $r_s$ and $r_w$, where all settings are the same as in Fig. 2.

The decoder function $S$ is also based on the inverse discrete Fourier transformation matrix as in [20], which allows successful detection of $b_2$ Byzantine ESs with probability 1.

**Step 2. Design of $\mathbf{c}^{(i)}$:** Next, we provide strategies to design $\mathbf{c}^{(i)}$, which allocates data partitions to the $i$-th ES. Suppose $2b_1+1$ divides $f$. Based on the data allocation vector $\mathbf{a}_i$, we divide the data partitions in the $i$-th cell into $\frac{f}{2b_1+1}$ *blocks* with equal size, having $\frac{k(2b_1+1)(2b_2+1)}{fL}$ data partitions in each block. Then from each block, we select the first $\frac{(2b_1+1)r_s}{f}$ data partitions to construct $\mathbf{c}^{(i)}$. The number of non-zero values of $\mathbf{c}^{(i)}$ becomes $\frac{(2b_1+1)r_s}{f} \times \frac{f}{2b_1+1} = r_s$. Here, the non-zero value of $\mathbf{c}^{(i)}$ in the $j$-th element is chosen to be the same as the value in the $j$-th column of $\mathbf{a}_i$. Fig. 3 shows an example with $n = k = 28$, $L = 4$, $b_1 = 1$, $b_2 = 1$. For each cell, the data partitions are divided into three data blocks containing seven data partitions each. In Fig. 3(a), the first three data partitions in each block are allocated to the ES, where the first five data partitions are allocated to the ES in Fig. 3(b). The non-zero elements of $\mathbf{c}^{(i)}$ can be obtained based on the matrix $\mathbf{A}$.

**Step 3. Design of $\mathbf{B}^{(i)}$ and $Q_i$:** Based on $\mathbf{a}_i$ and $\mathbf{c}^{(i)}$, we design $\mathbf{B}^{(i)}$ which allocates data partitions to the devices in cell $i$. As described in (4), the goal is to successfully obtain $\mathbf{a}_i\mathbf{G}$ at the $i$-th ES in the presence of $b_1$ Byzantine devices in the cell. Note that we have $\frac{k(2b_1+1)(2b_2+1)}{fL} - \frac{(2b_1+1)r_s}{f} = \frac{(2b_1+1)\{k(2b_2+1)-r_s L\}}{fL} = r_w^*$ remaining data partitions in each block after allocating data partitions to the ES in step 2. We call these remaining data partitions *subblocks*, where we have $\frac{f}{2b_1+1}$ types of subblocks in each cell. We replicate each type of subblock $2b_1+1$ times to have total of $f$ subblocks, and distribute across $f$ devices in the cell. All non-zero values of $\mathbf{B}^{(i)}$ are one. For the example in Fig. 3, each subblock in the $i$-th cell is replicated three times to generate a total of nine subblocks and allocated to the computing devices connected to the $i$-th ES (see Fig. 2).

Decoder function $Q_i$ takes the majority vote of the computed

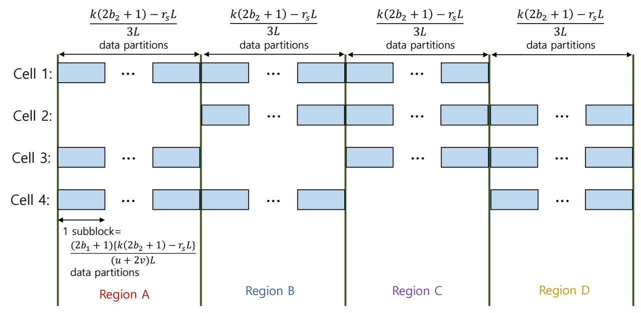

$\frac{k(2b_2+1)-r_sL}{3L}$ data partitions
$\frac{k(2b_2+1)-r_sL}{3L}$ data partitions
$\frac{k(2b_2+1)-r_sL}{3L}$ data partitions
$\frac{k(2b_2+1)-r_sL}{3L}$ data partitions
Cell 1:
Cell 2:
Cell 3:
Cell 4:
1 subblock= $\frac{(2b_1+1)\{k(2b_2+1)-r_sL\}}{(u+2v)L}$ data partitions
Region A
Region B
Region C
Region D

Fig. 4. Illustration of shared data partitions between cells, after allocating data to each ES.

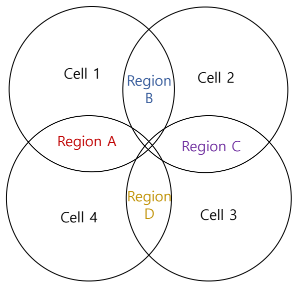


Fig. 5. Cell topology with $L = 4$.

results of the devices having the same type of subblock. Since there are $\frac{f}{2b_1+1}$ types of subblocks each replicated $2b_1 + 1$ times in each cell, $Q_i$ performs majority vote for $2b_1+1$ times. Hence, each cell $i$ is able to tolerate $b_1$ Byzantine nodes. After taking the majority vote for each type of subblock, the output is multiplied by a coefficient corresponding to the data partition of vector $\mathbf{a}_i$. For example, consider cell 1 in Fig. 2(b). For the devices having data partitions $D_{13}$ and $D_{14}$, after taking the majority vote, the result is multiplied by 2 which becomes a component of $2\tilde{\mathbf{g}}_2$. For the nodes having $D_{20}$ and $D_{21}$, we multiply the output of the majority vote by 3 to make the result to be a component of $3\tilde{\mathbf{g}}_3$.

Now due to the devices in the overlapping areas, joint design of $\{\mathbf{B}^{(i)}\}$ is required. As an example, consider a symmetric case as in Fig. 2 with $L = 4$ and $u_i = u$ for all $i \in [L]$, $v_{1,2} = v_{2,3} = v_{3,4} = v_{4,1} = v$, $v_{1,3} = v_{2,4} = 0$. According to (5) in Theorem 1, it is beneficial to increase $v$ (decrease $u$) as much as possible in reducing the computational load at the devices. However, if too many devices are connected to two ESs, it is not possible to jointly design $\{\mathbf{B}^{(i)}\}$ in a way consistent with the construction of $\mathbf{A}$ described earlier. If $u \geq v$ holds with $3(2b_1 + 1)|u + 2v$, we can always jointly design $\{\mathbf{B}^{(i)}\}$ to achieve the optimal bound. In this case, we first fill the first and last $v$ rows of $\{\mathbf{B}^{(i)}\}$, by allocating data partitions shared by adjacent cells. Then, we fill the remaining rows of $\{\mathbf{B}^{(i)}\}$ with the subblocks that are not allocated to the overlapping areas, which completes the construction. The details are as follows.

Case 1) If $b_2 = 0$, according to step 1 in code construction, there are no data partitions shared by adjacent cells. This means that there are no broadcasting devices ($v = 0$). Hence, it is always possible to achieve the optimal computational load at the devices.

Case 2) Now consider the case $b_2 = 1$. Suppose that $3(2b_1 + 1)$ divides $u + 2v$. This is equivalent to saying $\frac{(2b_1+1)\{k(2b_2+1)-r_sL\}}{(u+2v)L}$ divides $\frac{k(2b_2+1)-r_sL}{3L}$. Note that

$$\frac{(2b_1 + 1)\{k(2b_2 + 1) - r_sL\}}{(u + 2v)L} \tag{10}$$

is the number of data partitions contained in one subblock since $f = u+2v$. We also note that $\frac{k(2b_2+1)-r_sL}{L}$ types of data partitions are left at each cell after allocating $r_s$ data to each ES. Hence, if $\frac{(2b_1+1)\{k(2b_2+1)-r_sL\}}{(u+2v)L}$ divides $\frac{k(2b_2+1)-r_sL}{3L}$, we can always construct the subblocks as in Fig. 4. Now our goal is to allocate data partitions in the regions in Fig. 4 to the corresponding regions in Fig. 5, which shows the considered cell topology with $L = 4$.

Here, each region has $\frac{u+2v}{3(2b_1+1)}$ types of subblocks. We replicate each subblock $2b_1 + 1$ times and allocate to the devices in each overlapping cell region. Since there are $v$ devices in each overlapped region, if

$$(2b_1 + 1) \cdot \frac{u + 2v}{3(2b_1 + 1)} \geq v \tag{11}$$

holds, we can always construct a code by first allocating the shared subblocks to the overlapped region and then allocating the remaining subblocks to the non-overlapped area. The above equation (11) reduces to $u \geq v$, which completes the proof.

Case 3) If $b_2 \geq 2$, it is not possible to design a scheme for $L = 4$ since the majority of ESs are Byzantines. Hence, we do not consider this scenario.

Note that in practical setups, one can flexibly adjust $u$, $v$ values by connecting some devices in the overlapping region to only one ES (e.g., by allocating different frequency or time slots). While our design assumed the maximum numbers of Byzantines at both tiers ($b_1$ and $b_2$) are given, our method can easily handle the situation where Byzantine conditions are given in terms of the probability of a given ES or device being compromised. Depending on the maximum probability of a node being Byzantine at each tier, we can choose the code design parameters $b_1$, $b_2$ such that the probability of getting full gradient at the PS is greater than some target threshold.

In practice, the $u$, $v$ values may vary over time given potential mobility of the devices. How the mobility of the devices should be handled in the suggested scheme? PS can regularly be updated on missing/newcoming mobile devices and $u$, $v$ values by ESs. Based on the current status, PS and ESs can distribute newly encoded data partitions during downlink communications. Now, what if the location of a mobile node changes during a computing session? A simple solution is to rely on network connections among nearby ESs. Suppose all mobiles simply do the originally assigned jobs. If a mobile moves and gets associated with a new ES, then it simply informs the new ES of its old ES while passing on its calculation. The newly associated ES then simply passes that result along with the node ID to the old ES. The change from joint association to single association and vice versa can also be handled in a similar manner, assuming each node keeps track of its association status. All ESs will then get calculated gradients as planned. This would be a simple solution and will work since nearby ESs exchange information regularly even in existing networks via high-capacity links. The extra ES-to-ES traffic would be minimal here because only the calculated gradients (and node ID), not the original data, need be passed on.

### *C. Optimal Load Allocation*

So far, we designed a scheme achieving the fundamental bound on computational load to combat Byzantines, when $r_s$ is given. Now the question is how to choose appropriate $r_s$ and $r_w$ values, given certain amounts of computing powers at the devices and ESs. Let $P_w$ and $P_s$ be the computing power at the devices and ESs, respectively, meaning that computing gradients at each device/ES take $\frac{r_s}{P_s}$ and $\frac{r_w}{P_w}$ seconds, respectively. Heterogeneous computing powers across devices are not considered here. Under this setup, the following theorem provides the optimal load allocation solutions that minimize the running time of each iteration, which is the time taken from the beginning of the iteration until the PS decodes the exact sum of gradients.

**Theorem 2.** *For given $P_w$, $P_s$, optimal $r_s$ which minimizes the running time of each iteration is*

$$r_s = \frac{k(2b_1+1)(2b_2+1)}{\frac{P_w}{P_s}\sum_{i=1}^{L} f_i + L(2b_1+1)}. \tag{12}$$

*Optimal $r_w$ is obtained by inserting (12) to the right-hand side of (5) and taking the equality.*

*Proof:* Let $t_w$ be the communication time at each computing device for transmitting the aggregated gradient to the corresponding ES(s), which is affected by interference between signals sent from different devices. Similarly, we let $t_s$ be the communication time for transmitting the gradient from each ES to PS. When training begins, the ESs and devices compute gradients, which takes time of $\frac{r_s}{P_s}$ and $\frac{r_w}{P_w}$, respectively. Assuming that the communication from devices to ES(s) can be started after these gradient computations are finished, the running time of each iteration $T$ can be written as

$$T = \max\Big(\frac{r_s}{P_s}, \frac{r_w}{P_w}\Big) + t_w + T_w(f) + t_s + T_s(L). \tag{13}$$

Here, $T_w(f)$ and $T_s(L)$ are the decoding time at the ESs and the PS that are functions of $f$ and $L$, respectively. Now it can be seen that $T$ is minimized when

$$\frac{r_s}{P_s} = \frac{r_w}{P_w}. \tag{14}$$

By inserting

$$r_w = \frac{(2b_1+1)\{k(2b_2+1) - r_s L\}}{\sum_{i=1}^{L} f_i} \tag{15}$$

to (14), which is the minimum $r_w$ to guarantee Byzantines, we have $\frac{r_s}{P_s} = \frac{(2b_1+1)\{k(2b_2+1)-r_sL\}}{P_w\sum_{i=1}^{L} f_i}$. With some manipulations, this reduces to: $\{(2b_1+1)LP_s + P_w\sum_{i=1}^{L} f_i\}r_s = k(2b_1+1)(2b_2+1)P_s$. Finally, we obtain the optimal $r_s$ as follows:

$$r_s = \frac{k(2b_1+1)(2b_2+1)}{\frac{P_w}{P_s}\sum_{i=1}^{L} f_i + L(2b_1+1)}. \tag{16}$$

Optimal $r_w$ can be obtained by inserting (16) to (15), which completes the proof. ■

It can be seen from Theorem 2 that the optimal load at the ESs increases as $P_s$ increases or $P_w$ decreases. A telling observation is that optimal $r_s$ decreases as $\sum_{i=1}^{L} f_i$ increases, i.e., as we have more devices in the overlapping cell areas.

## IV. Experiments on Amazon EC2

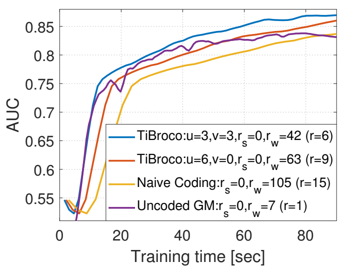
In this section, we provide experimental results tested on Amazon EC2. Here, we will refer to the computing devices, the ESs and the PS as the workers, the submasters and the master, respectively.

**Comparison schemes.** We compare TiBroco with the schemes that do not take the overlapping areas between cells into account ($u = \frac{n}{L}$, $v = 0$). First, we consider the most commonly used geometric median (GM) approach [16] with no coding, where the overall dataset is uniformly distributed across workers and submasters without replication, satisfying $k = Lr_s + nr_w$. Each submaster takes the GM of the collected results from the workers, combines it with the gradients computed by itself, and sends the result to the master. The master takes the GM of the collected results from the submasters and updates the global model. Secondly, we consider a naive coding scheme which does not consider the hierarchical nature; only the master decodes the result, not the submasters. The single-tier coding of [20] is directly applied from the master to the submasters and workers based on $r_s$, $r_w$. For a fair comparison, this code is designed targeting $b_1 + \frac{n}{L}b_2$ Byzantines out of $n$ workers. Mini-batch SGD with momentum is utilized for all schemes with a batch size of $k = 168$ and a momentum of 0.9. We define $r = \frac{Lr_s+nr_w}{k}$ as the *relative computational redundancy ratio* compared to the uncoded scheme.

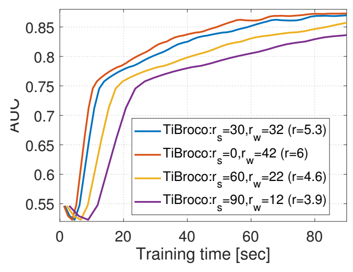
**Byzantine attack model.** Two attack models are considered. We consider the *reverse attack* where Byzantines flip the signs of true gradients. Each Byzantine sends $-c_1\mathbf{g}$ where $\mathbf{g}$ is the true gradient. We also consider *directional attack*, where the Byzantines send gradient of a certain direction. Instead of the true gradient, each Byzantine node sends $c_2\mathbf{1}_d$. We set $c_1 = c_2 = 100$ in our experiments.

**Experiments for logistic regression model.** Using the Amazon Employee Access dataset from Kaggle[3], we trained a logistic regression model with model size of $d = 263,500$. We used one-hot encoding to convert the categorical features to binary features as a preprocessing step. We utilized one master with `c4.2xlarge` instance, $L = 4$ submasters and $n = 24$ workers with `t2.micro` instances. At each iteration, we randomly selected $b_1 = 1$ worker and $b_2 = 1$ submaster to become Byzantines. As in Fig. 2, we assumed $u_i = u$ for all

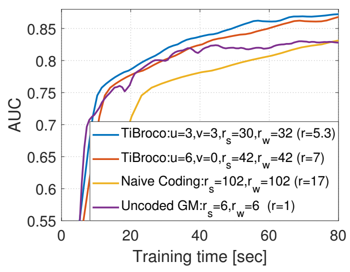
[3]https://www.kaggle.com/c/amazon-employee-access-challenge



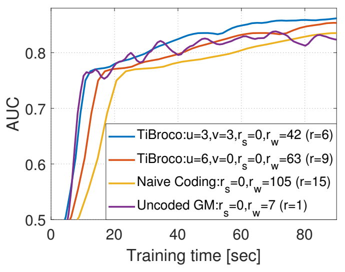
(a) $r_s = 0$, reverse attack



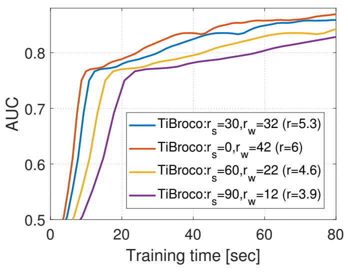
(b) $r_s \geq 0$, reverse attack



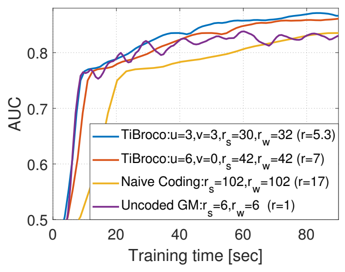
(c) $r_s$ optimized, reverse attack



(d) $r_s = 0$, directional attack



(e) $r_s \geq 0$, directional attack



(f) $r_s$ optimized, directional attack

Fig. 6. AUC vs time for training a logistic regression model ($r_s = 0$ means no computing at ES, $v = 0$ means no devices broadcasting to two ESs, and naive coding implies single-tier coding in [20]). Note that one can design a code for $u = 6$, $v = 0$ even when some devices are in the overlapping area, by connecting all devices in this overlapped region only to one BS. Similarly, one can design a code for $u = 3$, $v = 3$ even when more than 3 devices are in the overlapped region, by connecting only 3 devices in this region to both BSs.

$i \in [L]$ and $v_{1,2} = v_{2,3} = v_{3,4} = v_{4,1} = v$, $v_{1,3} = v_{2,4} = 0$, where $n = (u + v)L$ holds. This can be simply rewritten as $u + v = 6$ in our setup. For the uncoded GM scheme, each worker is responsible for $r_w = \frac{k-Lr_s}{n}$ data partitions. For our TiBroco, $r_w$ and $r_s$ are determined to achieve the fundamental bound in (5).

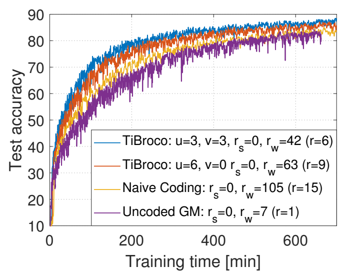
Fig. 6 shows AUC (area under curve) versus training time where 26,208 samples are used for training. We have several important observations from Fig. 6(a), which shows the performance of schemes with $r_s = 0$ (i.e., no gradient computation at ES) under reverse attack. First, the uncoded GM cannot reach the ideal AUC performance. More specifically, the uncoded GM approach cannot obtain the exact sum of gradients at each iteration, which degrades the AUC performance. Compared to uncoded GM, the coded schemes theoretically guarantee full gradient at the master to achieve the ideal AUC. This result is consistent with the plots in [20]. The second observation is the effect of hierarchical coding. Compared to TiBroco, the naive coding scheme (which does not reflect the hierarchical architecture) requires larger training time to reach the same AUC performance. Finally, it can be seen that the TiBroco which utilizes the devices in the overlapping cell region ($u = 3$, $v = 3$) provides better performance compared to the scheme which does not ($u = 6$, $v = 0$).

The impact of two-tier computing can be observed in Fig. 6(b), which plots the performance of scheme $u = 3$, $v = 3$ with different $r_s$ and $r_w$ values. All schemes can tolerate the same number of Byzantines with different loads at the submasters and workers. One can reduce $r_w$ by 1 by increasing $r_s$ by 3, which is consistent with the result (5) in Theorem 1. By comparing the schemes $(r_s, r_w) = (30, 32)$ and $(r_s, r_w) = (0, 42)$, it can be seen that the scheme that utilizes the computing powers of ESs can reduce the burden at the devices while combating the same number of Byzantines, which reduces the overall training time. Since we utilized the same type of machines for both submasters and workers, the training time is minimized when $\max\{r_s, r_w\}$ is minimized.

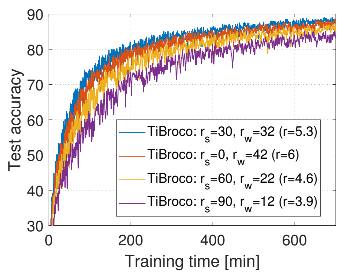
In Fig. 6(c), we plot the curves of each scheme with optimized $r_s$, $r_w$ values to minimize the average training time. The result clearly confirms the advantage of TiBroco which leverages hierarchical edge computing, hierarchical coding, and broadcasting wireless devices. Figs. 6(d), 6(e), 6(f) show the cases with directional attack, which are consistent with the results with reverse attack.

**Experiments for ResNet18.** We also confirm our results by training ResNet18 (model size of $d = 11,173,962$) with the CIFAR-10 dataset. The overall dataset is split into $50,000$ training samples and $10,000$ test samples. The learning rate is set to 0.01. We utilized `c4.2xlarge` instances for the master, submaster and workers, where the other settings are the same as in Fig. 6. Fig. 7 shows the test accuracy versus training time with CIFAR-10. The results are consistent with the plots in Fig. 6, indicating that TiBroco with theoretically optimal (minimal) computational load has significant advantages over existing uncoded GM method and naive coding.

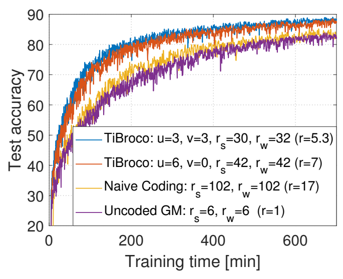
**Practical setup to reduce redundant computations.** We emphasize that our approach achieves the theoretical minimum bound on the required redundancy and thus is the most computationally-efficient scheme that is theoretically possible, *if* perfect Byzantine tolerance is to be targeted and *if* no compromise is allowed on the quality of the globally computed gradient. Having established that, we remark that there is



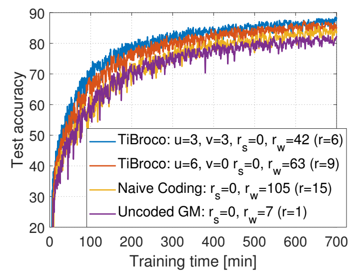
(a) $r_s = 0$, reverse attack



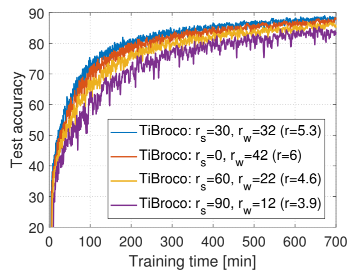
(b) $r_s \geq 0$ $(u = v = 3)$, reverse attack



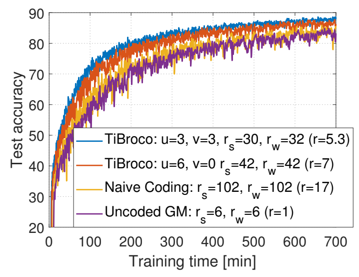
(c) $r_s$ optimized, reverse attack



(d) $r_s = 0$, directional attack



(e) $r_s \geq 0$ $(u = v = 3)$, directional attack



(f) $r_s$ optimized, directional attack

Fig. 7. Test accuracy versus time for training ResNet18 with CIFAR-10.

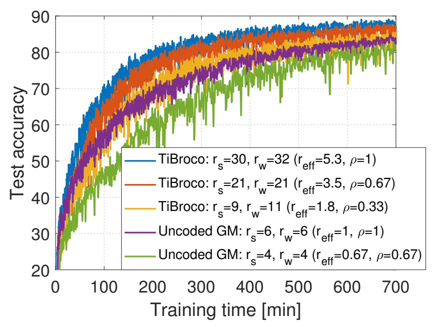


Fig. 8. Impact of reducing redundant computation for TiBroco with $u = 3$, $v = 3$. Test accuracy versus time is shown for training ResNet18 with CIFAR-10 under reverse attack. For all schemes, the cases with $\rho = 1$, $\rho = 0.67$, $\rho = 0.33$ correspond to mini-batch sizes of 168, 112, 56, respectively. It can be seen that TiBroco with a reduced effective redundancy $r_{\text{eff}}$ can still perform better than the uncoded scheme.

a simple way to lower computational burden by reducing the mini-batch size to $\rho k$ where $\rho < 1$ (and subsequent subsampling of the given data partitions by the same factor $\rho$), under the assumption that the resulting quality of the computed gradient can be negotiated. In this case, the effective relative redundancy ratio is given by $r_{\text{eff}} = \rho r$. Here we provide additional experimental results to show how the overall amount of redundant computation sometimes can be reduced so that Byzantine protection is still guaranteed with a controlled loss of quality in gradient computation. Fig. 8 shows the impact of reducing redundant computation for TiBroco with $u = 3$, $v = 3$, under reverse attack. We utilized CIFAR-10 for training ResNet18. For all schemes, the cases with $\rho = 1$, $\rho = 0.67$, $\rho = 0.33$ correspond to mini-batch sizes of 168, 112, 56, respectively. Although TiBroco loses performance as $\rho$ or $r_{\text{eff}}$ decreases, the overall results show that TiBroco with a significantly reduced computational load can still perform better than the uncoded GM, showing the effectiveness of the proposed coding idea. Also shown is the plot for the uncoded GM scheme with a reduced $\rho$ of $\rho = 0.67$ due to a reduced mini-batch size. It can be seen that the uncoded scheme also suffers performance degradation due to the reduced mini-batch size. The training time of each scheme achieving a given level of accuracy is more clearly seen in Table I. The overall results confirm the practical applicability of TiBroco, showing significantly better performance than the uncoded scheme while reducing redundant computations at the devices.

TABLE I
TRAINING TIME (MINUTES) ACHIEVING THE DESIRED TEST ACCURACY FOR RESNET18 IN FIG. 8. TIBROCO WITH $u = 3$, $v = 3$ IS UTILIZED.

| Target test accuracy | 75% | 80% | 85% |
|---|---|---|---|
| TiBroco ($r_{\text{eff}} = 5.3$, $\rho = 1$) | 133.7 | 191.9 | 342.7 |
| TiBroco ($r_{\text{eff}} = 3.5$, $\rho = 0.67$) | 154.4 | 240.0 | 373.9 |
| TiBroco ($r_{\text{eff}} = 1.8$, $\rho = 0.33$) | 250.8 | 354.3 | 682.4 |
| Uncoded GM ($r_{\text{eff}} = 1$, $\rho = 1$) | 287.1 | 409.1 | $\infty$ |
| Uncoded GM ($r_{\text{eff}} = 0.67$, $\rho = 0.67$) | 367.7 | 571.6 | $\infty$ |

## V. CONCLUSION

We proposed TiBroco, a coded distributed learning framework highly tailored to the hierarchical wireless edge networks. Our key ideas were to provide two-tier coding involving both ESs and the distributed computing devices. The code achieves the theoretically derived minimum bound on the computational redundancy in combating Byzantine attacks at both tiers. The code carefully exploits the presence of computing devices in the overlapping cell regions between nearby edge servers. Extensive experimental results show that the proposed TiBroco enables fast and secure distributed learning in practical tiered wireless edge networks plagued by Byzantine adversaries.

## References

[1] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang *et al.*, "Large scale distributed deep networks," in *Advances in neural information processing systems*, 2012, pp. 1223–1231.

[2] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, 2014, pp. 583–598.

[3] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Advances in Neural Information Processing Systems*, 2014, pp. 19–27.

[4] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent," in *Advances in Neural Information Processing Systems*, 2017, pp. 5330–5340.

[5] D. Datla, X. Chen, T. Tsou, S. Raghunandan, S. S. Hasan, J. H. Reed, C. B. Dietrich, T. Bose, B. Fette, and J.-H. Kim, "Wireless distributed computing: a survey of research challenges," *IEEE Communications Magazine*, vol. 50, no. 1, pp. 144–152, 2012.

[6] F. Wang, J. Xu, X. Wang, and S. Cui, "Joint offloading and computing optimization in wireless powered mobile-edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 17, no. 3, pp. 1784–1797, 2017.

[7] J. Ren, G. Yu, Y. Cai, and Y. He, "Latency optimization for resource allocation in mobile-edge computation offloading," *IEEE Transactions on Wireless Communications*, vol. 17, no. 8, pp. 5506–5519, 2018.

[8] K. Driscoll, B. Hall, H. Sivencrona, and P. Zumsteg, "Byzantine fault tolerance, from theory to reality," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2003, pp. 235–248.

[9] M. Vukolic, "The byzantine empire in the intercloud." *SIGACT News*, vol. 41, no. 3, pp. 105–111, 2010.

[10] M. Stanislav and T. Beardsley, "Hacking iot: A case study on baby monitor exposures and vulnerabilities," *Rapid 7*, 2015.

[11] M. Patton, E. Gross, R. Chinn, S. Forbis, L. Walker, and H. Chen, "Uninvited connections: a study of vulnerable devices on the internet of things (iot)," in *2014 IEEE Joint Intelligence and Security Informatics Conference*. IEEE, 2014, pp. 232–235.

[12] U. Acharya and M. Younis, "Increasing base-station anonymity in wireless sensor networks," *Ad Hoc Networks*, vol. 8, no. 8, pp. 791–809, 2010.

[13] U. Meyer and S. Wetzel, "A man-in-the-middle attack on umts," in *Proceedings of the 3rd ACM workshop on Wireless security*. ACM, 2004, pp. 90–97.

[14] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *International Conference on Machine Learning*, 2018, pp. 5650–5659.

[15] ——, "Defending against saddle point attack in byzantine-robust distributed learning," in *International Conference on Machine Learning*. PMLR, 2019, pp. 7074–7084.

[16] Y. Chen, L. Su, and J. Xu, "Distributed statistical machine learning in adversarial settings: Byzantine gradient descent," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 1, no. 2, p. 44, 2017.

[17] P. Blanchard, R. Guerraoui, J. Stainer *et al.*, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Advances in Neural Information Processing Systems*, 2017, pp. 119–129.

[18] C. Xie, O. Koyejo, and I. Gupta, "Generalized byzantine-tolerant sgd," *International Conference on Machine Learning*, 2018.

[19] E. M. El Mhamdi, R. Guerraoui, and S. L. A. Rouault, "The hidden vulnerability of distributed learning in byzantium," in *International Conference on Machine Learning*, no. CONF, 2018.

[20] L. Chen, H. Wang, Z. Charles, and D. Papailiopoulos, "Draco: Byzantine-resilient distributed training via redundant gradients," *International Conference on Machine Learning*, 2018.

[21] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*.

[22] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[23] D.-J. Han, J.-y. Sohn, and J. Moon, "Hierarchical broadcast coding: Expediting distributed learning at the wireless edge," *IEEE Transactions on Wireless Communications (Early Access), DOI: 10.1109/TWC.2020.3040792*, 2020.

[24] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, 2017.

[25] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," in *Advances in Neural Information Processing Systems*, 2017, pp. 4403–4413.

[26] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *International Conference on Machine Learning*, 2017, pp. 3368–3376.

[27] M. Ye and E. Abbe, "Communication-computation efficient gradient coding," in *International Conference on Machine Learning*, 2018.

[28] N. Raviv, I. Tamo, R. Tandon, and A. G. Dimakis, "Gradient coding from cyclic mds codes and expander graphs," in *International Conference on Machine Learning*, 2018.

[29] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Tree gradient coding," in *2019 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2019, pp. 2808–2812.

[30] D.-J. Han, J.-y. Sohn, and J. Moon, "Coded distributed computing over packet erasure channels," in *2019 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2019, pp. 717–721.

[31] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security and privacy," *arXiv preprint arXiv:1806.00939*, 2018.

[32] J.-y. Sohn, D.-J. Han, B. Choi, and J. Moon, "Election coding for distributed learning: Protecting signsgd against byzantine attacks," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[33] S. Rajput, H. Wang, Z. Charles, and D. Papailiopoulos, "Detox: A redundancy-based framework for faster and more robust gradient aggregation," in *Advances in Neural Information Processing Systems*, 2019, pp. 10 320–10 330.