Election Coding for Distributed Learning: Protecting SignSGD against Byzantine Attacks

Jy-yong Sohn, Student Member, IEEE, Dong-Jun Han, Student Member, IEEE, Beongjun Choi, Student Member, IEEE, and Jaekyun Moon, Fellow, IEEE

Abstract

Recent advances in large-scale distributed learning algorithms have enabled communication-efficient training via SignSGD. Unfortunately, a major issue continues to plague distributed learning: namely, Byzantine failures may incur serious degradation in learning accuracy. This paper proposes Election Coding, a coding-theoretic framework to guarantee Byzantine-robustness for SignSGD with Majority Vote, which uses minimum worker-master communication in both directions. The suggested framework explores new information-theoretic limits of finding the majority opinion when some workers could be malicious, and paves the road to implement robust and efficient distributed learning algorithms. Under this framework, we construct two types of explicit codes, random Bernoulli codes and deterministic algebraic codes, that can tolerate Byzantine attacks with a controlled amount of computational redundancy. For the Bernoulli codes, we provide upper bounds on the error probability in estimating the majority opinion, which give useful insights into code design for tolerating Byzantine attacks. As for deterministic codes, we construct an explicit code which perfectly tolerates Byzanitines, and provide tight upper/lower bounds on the minimum required computational redundancy. Finally, the Byzantine-tolerance of the suggested coding schemes is confirmed by deep learning experiments on Amazon EC2 using Python with MPI4py package.

I. INTRODUCTION

The modern machine learning paradigm is moving toward parallelization and decentralization for providing fast and efficient solutions to complex real-world problems, which involve training high-dimensional network models using massive data. There has been extensive work on
developing distributed learning algorithms [4]–[9] to exploit large-scale computing units. These distributed learning algorithms are usually implemented in parameter-server (PS) framework [10], where a central PS (or master) aggregates the computational results (e.g., gradient vector which minimizes empirical loss) of distributed workers to update the shared model parameters. In recent years, two issues have emerged as major drawbacks that limit the performance of distributed learning: Byzantine failures and communication burden.

Byzantine nodes send completely arbitrary messages to PS, which mislead the model updating process and severely degrade learning capability. In order to counter the threat of Byzantine attacks, much attention has been focused on robust solutions [11]–[13]. Motivated by the fact that even a single Byzantine node cannot be tolerated by using naive linear aggregation rules at PS, the authors of [14]–[16] considered median-based aggregation rules. However, as data volume and the number of workers increase, taking the median involves a huge computational cost [15] which is far greater than the cost for batch gradient computations. Thus, another work [17] instead suggested redundant gradient computation that tolerates arbitrary attacks by Byzantines.

Another issue is high communication burden caused by transmitting gradient vectors between PS and workers for updating network models. Regarding this issue, the authors of [18]–[24] considered quantization of real-valued gradient vectors. The signed stochastic gradient descent method (SIGNSGD) suggested in [21] compresses a real-valued gradient vector $g$ into a binary vector $\text{sign}(g)$, and updates the model using the 1-bit compressed gradients. This scheme minimizes the communication load from PS to each worker for transmitting the aggregated gradient. A further variation called SIGNSGD WITH MAJORITY VOTE (SIGNSGD-MV) [21], [22] also applies 1-bit quantization on gradients communicated from each worker to PS in achieving minimum master-worker communication in both directions. Moreover, a recent work [25] suggested feedback for compensating the error caused by the biased gradient compression in SIGNSGD. These schemes have been shown to minimize the communication load while maintaining the SGD-level convergence speed in general non-convex problems. A major issue that remains, however, is the lack of Byzantine-robust solutions suitable for such communication-efficient learning algorithms.

**Main Contributions:** In this paper, we propose ELECTION CODING, a coding-theoretic framework for making SIGNSGD-MV [21] highly robust to Byzantine attacks. In particular, we focus on estimating the next step $\mu$ for model update, i.e., the majority voting on the signed gradients extracted from $n$ data partitions distributed across a network, under the assumption that
of the $n$ worker nodes are under arbitrary Byzantine attacks. With our ELECTION CODING, we assign each data partition to multiple workers, and we show that this redundant data allocation and corresponding redundant gradient computation enable accurate estimation on $\mu$ under Byzantine failures. In the context of voting systems, ELECTION CODING explores coding opportunities for estimating the majority opinion $\mu$ of multiple voters, where each vote could be tampered with by an adversary act.

At the more specific level, we construct two coding schemes: random Bernoulli codes and algebraic deterministic codes. Regarding the random Bernoulli codes, which are based on arbitrarily assigning data partitions to each worker node with (connection) probability $p$, we obtain upper bounds on the error probability in estimating $\mu$. This provides a guideline for selecting $p$ depending on the number of Byzantine nodes. As for the deterministic codes, we first obtain the necessary and sufficient condition on the data allocation rule, in order to accurately estimate $\mu$ under Byzantine attacks. Afterwards, we suggest an explicit coding scheme which achieves the perfect Byzantine tolerance for arbitrary $n, b$ values, with $n$ being the total number of nodes. We also provide tight upper/lower bounds on the minimum required computational redundancy $r^*$ for perfect Byzantine tolerance.

Finally, the mathematical results are also confirmed by simulations on well-known machine learning architectures. We implement the suggested coded distributed learning algorithms in PyTorch, and deploy them on Amazon EC2 using Python with MPI4py package. We trained RESNET-18 using CIFAR-10 dataset as well as a logistic regression model using Amazon Employee Access dataset from Kaggle. The experimental results confirm that the suggested coded algorithm requires significantly less training time to achieve a target test accuracy compared to the uncoded case, under different types of attacks.

Related Works: The authors of [17] suggested a coding-theoretic framework DRACO for Byzantine-robustness of distributed learning algorithms. However, compared to the codes in [17], the codes suggested in this paper are more suitable for SIGNEDSGD setup (or in general compressed gradient schemes), due to the following two advantages. First, our codes have much smaller encoding/decoding complexities than the codes in [17]. At each mini-batch iteration in the training phase, the codes proposed in [17] require multiplying real-valued (or complex-valued) coefficients to the gradient vector for encoding (or decoding), while our codes require a simple majority vote operation on the binary elements in both encoding and decoding. Second, the
probabilistic Bernoulli random codes suggested in this paper can be designed in a more flexible manner. The codes in [17] require $r = 2b + 1$ computational redundancy, where $b$ is the number of Byzantines. Thus, the required redundancy increases in a linear function with $b$, which is burdensome for large $b$. The probabilistic codes suggested in this paper can be designed in a more flexible manner to tolerate Byzantines: we can control the redundancy by choosing an appropriate connection probability $p$. Simulation results show that our codes having the expected redundancy of $E[r] = 2$ enjoy significant gain compared to the uncoded scheme when $n = 49$ and $b = 5$, while the codes in [17] require the redundancy of $r = 11$. A recent work [26] suggested a framework DETOX which combines two existing schemes: computing redundant gradients and applying robust gradient aggregation methods. However, DETOX still suffers from a high computational overhead compared to our scheme, since it is based on a robust aggregation scheme, e.g., geometric median aggregator.

For communicating 1-bit compressed gradients, a recent work [22] analyzed the Byzantine-tolerance of the naive SIGNSGD-MV scheme. This scheme can only achieve a limited accuracy as the number of Byzantines $b$ increases, whereas the proposed coding scheme can achieve the ideal accuracy of $b = 0$ scenario, regardless of the number of Byzantines. This difference can be observed in the simulation results provided in Section V. We note that the suggested scheme reduces to the naive scheme in [22] when the data allocation matrix is an identity matrix.

The role of codes in distributed learning systems has been investigated widely in the literature. Extensive work has focused on exploiting codes for reducing the runtime of learning algorithms in the presence of straggling worker nodes [27]–[35]. The authors of [17] suggested codes for Byzantine-resilient distributed learning system. However, none of these works suggest codes to combat Byzantine attacks for the communication-efficient SIGNSGD-MV algorithms.

Another related topic is making a collective decision from noisy observations on voters’ opinions, which is considered in the area of social choice and ranking system. Some previous works [36], [37] related the estimation of the majority vote with an error correction process. These previous works considered each vote $m_i$ as a noisy perception of repetition-coded version of the ground-truth majority opinion $\mu$, and devised estimation techniques for such a scenario. This system setting is fundamentally different from that of the present work, which considers a two-step encoding/decoding scenario where each worker gathers opinions of a subset of voters, and the master decides the majority vote based on the observations from workers.
II. SYSTEM MODEL

A. Distributed Learning using SIGNSGD with Majority Vote (SIGNSGD-MV)

Here we review distributed learning algorithms which use SIGNSGD with Majority Vote (SIGNSGD-MV). Consider a distributed learning system using $n$ workers. We divide the training data into $n$ partitions, denoted as $\{D_i\}_{i \in [n]}$. The gradient vector computed from data partition $D_i$ is denoted as $g_i = [g_{i,1}, g_{i,2}, \cdots, g_{i,d}]$ where $d$ is the dimension of parameter space $\Omega$. For a specific coordinate $l \in [d]$, the set of gradient elements computed for $n$ data partitions is denoted as $g^{(l)} = [g_{1,l}, g_{2,l}, \cdots, g_{n,l}]$. A message vector $m^{(l)} = [m_{1,l}, m_{2,l}, \cdots, m_{n,l}]$ where $m_{i,l} \in \{0, 1\}$. We also define the majority opinion as $\mu^{(l)} = \text{maj}(m^{(l)})$, where $\text{maj}(\cdot)$ is a majority function which outputs the more frequent element in the input argument vector. We update the model parameter as $\omega_{s+1} = \omega_s + \gamma \mu$, where $\gamma$ is the learning rate and $\mu = [\mu^{(1)}, \cdots, \mu^{(d)}]$. 

Figure 1: System model for estimating the majority opinion $\mu$ in the suggested ELECTION CODING framework. This framework is applied for each coordinate of the model parameter $\omega_s \in \Omega$ in a parallel manner.

Notations: The sum of elements of vector $v$ is denoted as $\|v\|_0$. Similarly, $\|M\|_0$ represents the sum of elements of matrix $M$. An $n \times n$ identity matrix is denoted as $I_n$. The set $\{1, 2, \ldots, n\}$ is denoted by $[n]$. An $n \times k$ all-one matrix is denoted as $\mathbb{1}_{n \times k}$. For a given set $S$, the identification function $\mathbb{1}_{\{x \in S\}}$ outputs one if $x \in S$, and outputs zero otherwise.
The suggested ELECTION CODING framework for estimating the majority opinion \( \mu^{(l)} \) is illustrated in Fig. 1a. Since we consider coordinate-wise encoding and decoding, we focus only on one dimension; we shall drop the index \( l \). The binary message vector \( m^{(l)} \) is now simply denoted as \( m = [m_1, \cdots, m_n] \), and the majority opinion \( \mu^{(l)} \) as \( \mu = \text{maj}(m) \). This paper suggests applying codes for allocating data partitions into worker nodes. We assume that \( n \) is an odd number, in order to avoid ambiguity at the output of the majority function. We define data allocation matrix \( G \in \{0, 1\}^{n \times n} \) as follows: \( G_{ji} = 1 \) if data partition \( i \) is allocated to node \( j \), and \( G_{ji} = 0 \) otherwise. Then, we define \( P_j = \{i \in [n] : G_{ji} = 1\} \), the set of data partitions assigned to node \( j \). Given a data allocation matrix \( G \), the computational redundancy compared to the naive uncoded scheme is expressed as \( r = \|G\|_0/n \), the average number of data partitions handled by each node. Note that the uncoded scheme corresponds to \( G = I_n \). Once node \( j \) computes \( \{m_i\}_{i \in P_j} \) from the assigned data partitions, it generates a binary information \( c_j = E_j(m; G) \) using encoder \( E_j \). We use the notation \( c = [c_1, \cdots c_n] \) for \( n \) bits generated by the worker nodes.

After generating \( c_j \in \{0, 1\} \), node \( j \) transmits\(^1\)

\[
y_j = \begin{cases} 
\mathcal{X}, & \text{if node } j \text{ is a Byzantine} \\
c_j, & \text{otherwise}
\end{cases}
\]  

(1)

to PS, where \( \mathcal{X} \) is either \( c_j \oplus 1 \) or \( c_j \) since each node is allowed to transmit either 0 or 1. Following the related work [17], [22], we assume that the number of Byzantine nodes satisfy \( b \in \{0, 1, \cdots, [n/2]\} \). The maximum number of Byzantines is denoted as \( b_{\text{max}} = [n/2] \). After an arbitrary attack of \( b \) Byzantines, PS observes \( y = [y_1, y_2, \cdots, y_n] \) and estimates \( \mu \) using a decoding function \( D : y \mapsto \hat{\mu} \).

In Fig. 1b, we illustrate an example of the suggested framework used for a voting scenario where \( n \) voters vote for either bit 1 or bit 0. Each polling station observes the votes corresponding to some subset of voters. Assume that each polling station must send the central election commission a single bit most representative of its local votes. A natural choice is to find the majority. Some

\(^1\)Since a Byzantine node behaves arbitrarily, it may transmit nothing, or it may be a straggling node. Note that ELECTION CODING is also tolerant to this scenario; when a coding scheme guarantees the master to correctly estimate the majority opinion \( \mu \) under bit flip attacks, it trivially guarantees the correct estimate at the master under bit erasure scenarios as well. In the example of Fig. 1b, the master successfully obtains \( \hat{\mu} = 0 \) even when the Byzantine node transmit nothing instead of sending wrong information \( y_5 = 1 \).
polling stations may turn out to be Byzantines, arbitrarily changing the voting results. The master wishes to estimate the majority vote of the original $n$ voters by observing the majority of the majority votes compiled by $n$ polling stations, some of which may be Byzantines. The example in the figure shows that although a Byzantine station flips a bit, the master can still accurately estimate $\mu$. In this example, coding amounts to telling each voter to go to which polling stations. By sending each voter to multiple stations in some predefined way, the voting system becomes resistant to Byzantine attacks to change the majority voting results of the polling stations.

C. Target Problem

Coming back to the distributed gradient computation problem, there are three key system design parameters which affect the accuracy in estimating $\mu$: the task allocation matrix $G$, the encoder functions $\{E_j\}_{j \in [n]}$ at $n$ worker nodes, and the decoder function $D$ at the master. In this paper, we focus on low-complexity hierarchical voting where both the encoder and the decoder are majority voting functions:

\begin{equation}
    c_j = E_j(m; G) = \text{maj}(\{m_i\}_{i \in P_j})
\end{equation}

\begin{equation}
    \hat{\mu} = D(y) = \text{maj}(y_1, y_2, \cdots, y_n).
\end{equation}

Under this setting, we define the Byzantine tolerance of a given system as follows.

**Definition 1.** Consider a system having a data allocation matrix $G$, encoders $\{E_j\}_{j \in [n]}$ in (2), and a decoder $D$ in (3). We define the system to be $(b, \epsilon)-$Byzantine tolerable, if it can tolerate any types of attack from $b$ Byzantine nodes with at least probability $1 - \epsilon$, or equivalently, $P(\hat{\mu} \neq \mu) \leq \epsilon$ for arbitrary attack scenarios. For the case where estimation error is zero, $\epsilon = 0$, we say that the system is perfect $b-$Byzantine tolerable.

This paper focuses on achieving $(b, \epsilon)-$Byzantine tolerance by optimally using system resources of computation and communication. Specifically, we ask the following key question: Assuming minimum worker-master communication in both directions (i.e., under SINGSGD-MV), in order to tolerate arbitrary attacks from $b$ Byzantine nodes with at least probability $1 - \epsilon$, how should we design the data allocation matrix $G$ by using the minimum redundant computation?
III. SYSTEM DESIGN USING RANDOM BERNOUlli CODES

We first suggest random Bernoulli codes, where each node randomly selects each data partition with connection probability \( p \) independently. Then, \( \{G_{ji}\} \) are i.i.d Bernoulli random variables with \( G_{ji} \sim \text{Bern}(p) \). The idea of randomly contacting messages at each coded bit has been considered in previous work on fountain codes \cite{38} and Bernoulli gradient codes \cite{35}. However, using this idea for tolerating Byzantines in distributed learning is something entirely different and requires unique analysis. Note that depending on the Byzantine attack scenario, flexible code construction is available by adjusting the connection probability \( p \).

A. Estimation Error Bound for Random Bernoulli Codes

For a random Bernoulli code, the error probability \( P(\hat{\mu} \neq \mu) \) of estimating the majority value \( \mu \) is bounded as follows.

**Theorem 1.** Consider assigning data partitions into \( n \) nodes using data allocation matrix \( G \) generated by i.i.d. Bernoulli random variables with \( G_{ji} \sim \text{Bern}(p) \) for some \( p > 0 \). Let \( b \) be the number of Byzantine nodes. Then, in the asymptotic regime of large \( n \), the system is \((b, \delta)\)-Byzantine tolerable, i.e., the error probability is upper bounded as

\[
P(\mu \neq \hat{\mu}) \leq \delta
\]

where \( \delta \) is the probability of having more than \( \lfloor n/2 \rfloor - b \) nodes outputting wrong estimates on the majorities, which can be expressed as

\[
\delta = \frac{1}{2^{n-1}} \sum_{w=1}^{\lfloor n/2 \rfloor} \binom{n}{w} \sum_{s=\lfloor n/2 \rfloor - b + 1}^{n} \binom{n}{s} q_{w}^{s} (1 - q_{w})^{n-s} + o(1) \tag{4}
\]

with

\[
q_{w} = \sum_{v=1}^{2w} \sum_{i=\lfloor v/2 \rfloor}^{\min\{w,v\}} \binom{w}{i} \binom{n-w}{v-i} p^{i} (1-p)^{n-v}. \tag{5}
\]

**Proof.** The full proof is given in Appendix B here we just provide a sketch. Consider an arbitrary message vector \( m \) having weight \( \|m\|_{0} = \omega \leq \lfloor n/2 \rfloor \). Then, the majority opinion is \( \mu = 0 \).

\[\text{For message vectors with } \|m\|_{0} > \lfloor n/2 \rfloor, \text{ a similar approach gives us the same result; the only difference is } \mu = 1.\]
Thus, the estimation error event occurs when more than \([n/2]\) nodes transmit \(y = 1\), resulting in \(\hat{\mu} = 1\). We first obtain \(q_w\), the probability of a given node outputting the computational result \(c = 1\). Note that \(q_w\) is the same for all nodes, since \(\{G_{ji}\}\) are i.i.d. Bernoulli random variables.

For an arbitrary realization of \(G\), suppose \(v\) data partitions are allocated to a given node. Then, the node outputs \(c = 1\) when more than \([v/2]\) partitions have message \(m = 1\). The probability of this event is expressed using a combinatorial term in (5). Recall that, as state above, the estimation error event (\(\mu \neq \hat{\mu}\)) occurs when more than \([n/2]\) nodes transmit \(y = 1\). In the worst case of having \(y = 1\) for all \(b\) Byzantine nodes, the estimation error probability \(\Pr(\hat{\mu} \neq \mu|m)\) reduces to the probability of having more than \([n/2] - b\) nodes with \(c = 1\). In other words,

\[
\Pr(\hat{\mu} \neq \mu|m) \leq \Pr(|\{j \in [n]: c_j = 1\}| \geq [n/2] - b + 1) = \sum_{s=[n/2]-b+1}^{n} \binom{n}{s} q_w^s (1-q_w)^{n-s}. \tag{6}
\]

holds. Taking the weighted sum of these terms for various \(m\) results in \(\delta\) in (4).

The error bound expression for \(\delta\) as given in (4) is a bit too complicated to develop useful insights. We provide a rougher but simpler bound \(\epsilon\) in the following corollary that would provide better physical interpretations on the behavior of the suggested random Bernoulli codes.

**Corollary 1.** Consider using random Bernoulli codes with \(G_{ji} \sim \text{Bern}(p)\) for some \(p > 0\). Let \(b\) be the number of Byzantine nodes. Then, in the asymptotic regime of large \(k\), the system is \((b, \epsilon)\)-Byzantine tolerable, i.e., the error probability is upper bounded as

\[
\Pr(\hat{\mu} \neq \mu) \leq \epsilon
\]

where

\[
\epsilon = \frac{1}{2^{n-1}} \frac{n}{[n/2] - b + 1} \sum_{w=1}^{[n/2]} \binom{n}{w} q_w + o(1) \tag{7}
\]

and \(q_w\) is as in (5).

**Proof.** Recall that the conditional estimation error \(\Pr(\hat{\mu} \neq \mu|m)\) for a given message vector \(m\) is bounded as in (6). Using the Markov inequality, we have

\[
\Pr(\hat{\mu} \neq \mu|m) \leq \frac{\mathbb{E}[|\{j \in [n]: c_j = 1\}|]}{[n/2] - b + 1} = \frac{nw}{[n/2] - b + 1}.
\]
Taking the weighted sum of these terms for various message vectors $m$ results in $\epsilon$ in (7).

The error probability bound in Corollary 1 provides some physical intuition about how vulnerable a community with $n$ nodes is to the attack of $b$ Byzantine nodes. To be specific, the theorem relates two probabilities: $P(\hat{\mu} \neq \mu)$ which represents the probability that the master (aggregating the opinions of $n$ workers) makes a wrong decision on the majority value, and $P(c \neq \mu)$ which represents the decision error probability of an individual node. As explained in the proof of Theorem 1, $q_w$ is the probability of a non-Byzantine node outputting a wrong decision $c \neq \mu$ on the majority value, for a given message vector $m$ with $\|m\|_0 = w$. Thus,

$$P(c \neq \mu) := \frac{1}{2n} \sum_{m} P(c \neq \mu|m) \stackrel{(a)}{=} \frac{1}{2^{n-1}} \sum_{\|m\|_0 \leq [n/2]} P(c \neq \mu|m) = \frac{1}{2^{n-1}} \sum_{w=1}^{[n/2]} \binom{n}{w} q_w$$

is the decision error probability of a given node, when $2^n$ message vectors $m$ are generated with equal probabilities. Here, (a) is from the fact that the error analysis for message vectors $m$ satisfying $\|m\|_0 > [n/2]$ is the same as the analysis for message vectors with $\|m\|_0 \leq [n/2]$, as explained in footnote 2. Now, the result of Theorem 1 can be written as

$$P(\hat{\mu} \neq \mu) \leq P(c \neq \mu) \frac{n}{[n/2] - b + 1} \simeq P(c \neq \mu) \left( \frac{1}{2} - \frac{b}{n} \right)^{-1}. \tag{8}$$

This implies that the probability $P(\hat{\mu} \neq \mu)$ of the community making a wrong decision is no more than $\frac{n}{[n/2] - b + 1}$ times the decision error probability $P(c \neq \mu)$ of an individual node. Note that the scaling factor is a function of how far $\frac{b}{n}$ is from its maximum value $\frac{1}{2}$.

B. Behavior of Majority Estimation Error

The error upper bound $\delta$ in Theorem 1 as well as the simulated error $P(\mu \neq \hat{\mu})$ are shown in Fig. 2a, when $n = 49$. We can check that both $\delta$ and $P(\mu \neq \hat{\mu})$ decrease as the connection probability $p$ increases, or equivalently, as the data allocation matrix $G$ becomes more dense. This makes sense because as $p$ increases, each node gets access to more data partitions on average, so that the probability of an honest (non-Byzantine) node correctly estimating the majority opinion $\mu$ increases. This decreases the estimation error regardless of the behavior of Byzantines. The cost we need to pay for this is the increased expected computational redundancy (and the download traffic for allocating data partitions) of $E[r] = E[\|G\|_0] / n = np$. One can get some guidance for designing random Bernoulli codes from the plot of upper bound $\delta$. For example, when there
are $b = 3$ Byzantines, it is necessary to set connection probability $p \geq 0.15$ to guarantee that estimation error is less than $30\%$. Now, in the following corollary, we analyze the behavior of error bound $\epsilon$ as the portion of Byzantine nodes $\alpha = b/n$ varies. Note that Corollary 2 is directly obtained from Corollary 1 by inserting $b = n\alpha$.

**Corollary 2.** Let $\alpha$ be the portion of Byzantine nodes, i.e., the number of Byzantine nodes is $b = n\alpha$ for some $\alpha \in (0, 1/2)$. Then, in the asymptotic regime of large $n$, the error bound $\epsilon$ in (7) is inversely proportional to $(1/2 - \alpha)$, i.e.,

$$\epsilon \propto \frac{1}{1/2 - \alpha}.$$  \hspace{1cm} (9)

Corollary 2 states that the error bound $\epsilon$ is inversely proportional to $(1/2 - \alpha)$ as the portion of Byzantines $\alpha = b/n$ varies. A very similar behavior is observed for the simulated error $P(\mu \neq \hat{\mu})$, as shown in Fig. 2b. We can confirm that the estimation error increases as $\alpha$ increases, and it increases faster as the portion of Byzantines $\alpha$ approaches its maximum towards $1/2$.

**IV. System Design Using Deterministic Codes**

Here we construct codes for *perfect* $b$–Byzantine tolerance, i.e., codes that tolerate any attacks from $b$ Byzantine nodes with probability 1, when the entries of $G$ are fixed. We use the notation $G(j,:)$ to represent $j^{th}$ row of matrix $G$. We assume that the number of data partitions $|P_j| = \|G(j,:)\|_0$ assigned to each node $j$ is an odd number, to avoid ambiguous output of the majority function in (2). For a given message vector $m = [m_1, m_2, \cdots, m_n]$, we define

$$J_v(m) := \{ j \in [n] : m^T G(j,:) \geq v, \|G(j,:)\|_0 = 2v - 1 \},$$  \hspace{1cm} (10)
which is the set of nodes with at least $v$ partitions having messages $m_i = 1$, out of $2v - 1$ allocated data partitions. Since each node takes the majority vote, we have $c_j = \text{maj}(\{m_i\}_{i \in P_j}) = 1$ for $j \in J_v(m)$. Under this setting, we define $r^*$, the minimum required computational redundancy for perfect Byzantine tolerance.

**Definition 2.** Consider using a deterministic data allocation matrix $G$ for $n$ nodes. The minimum redundancy required for perfect $b$–Byzantine tolerance is defined as

$$r^*(n, b) := \min_{G \in \{0, 1\}^{n \times n}} \frac{\|G\|_0}{n}$$

s.t. $(G, \{E_j\}_{j \in [n]}, D)$ is perfect $b$ – Byzantine tolerable,

where encoder $E$ and decoder $D$ are defined in (2) and (3), respectively.

**A. Code Constructions for Perfect $b$–Byzantine Tolerance**

In this section, we provide data allocation matrices $G$ that satisfy the perfect $b$–Byzantine tolerance. To begin, we provide the necessary and sufficient condition on $G$ to tolerate $b$ Byzantines in a perfect manner.

**Theorem 2.** Consider using a deterministic (non-random) data allocation matrix $G$. Then, the system is perfect $b$–Byzantine tolerable if and only if

$$\sum_{v=1}^{\lfloor n/2 \rfloor} |J_v(m)| \leq \left\lfloor \frac{n}{2} \right\rfloor - b$$

for all message vectors $m$ having weight $\|m\|_0 = \lfloor n/2 \rfloor$.

**Proof.** The formal proof is in Appendix C and here we just provide an intuitive sketch. Recall that the majority opinion is $\mu = 0$ when the message vector $m$ has weight $\|m\|_0 \leq \lfloor n/2 \rfloor$. Moreover, in the worst case attacks from $b$ Byzantines, the output $y_j$ and the computational result $c_j$ of node $j$ satisfy $J_0 := |\{j : y_j = 1\}| = |\{j : c_j = 1\}| + b$. Since the estimate on the majority opinion is $\hat{\mu} = \text{maj}\{y_1, \cdots, y_n\}$, the sufficient and necessary condition for accurate estimation (i.e., $\hat{\mu} \neq \mu$) is $J_0 \leq \lfloor n/2 \rfloor$, or equivalently, $|\{j : c_j = 1\}| = \sum_{v=1}^{\lfloor n/2 \rfloor} |J_v(m)| \leq \left\lceil \frac{n}{2} \right\rceil - b$. 

Using the result of Theorem 2 which specifies the condition for perfect $b$–Byzantine tolerance, we now construct explicit matrices $G$ that guarantee perfect $b$–Byzantine tolerance, under various
Algorithm 1: Data allocation matrix $G$ satisfying perfect $b$–Byzantine tolerance ($0 < b < b_{\text{max}}$)

**Input:** Number of nodes $n$, number of Byzantine nodes $b$.

**Output:** Data allocation matrix $G \in \{0, 1\}^{n \times n}$ that achieve the perfect $b$–Byzantine tolerance.

**Initialize:** Define $s = \frac{n-1}{2} - b$ and $L = \lfloor \frac{n-(2b+1)}{2(b+1)} \rfloor + 1$. Initialize $G$ as the all-zero matrix.

**Step 1:** Set the top left $s$-by-$s$ submatrix of $G$ as the identity matrix, i.e., $G(1:s, 1:s) = I_s$.

**Step 2:** Set the bottom $(n-s-L)$ rows as the all-one matrix, i.e., $G(s+L+1:n, :) = 1_{(n-s-L) \times n}$.

**Step 3:** Fill in the matrix $A := G(s+1:s+s+L, s+1:n)$ as follows: Insert $2b+1$ ones on each row by shifting the location by $b+1$, i.e., $A(l, (l-1)(b+1) + (1:2b+1)) = 1_{1 \times (2b+1)}$ for $l = 1, \ldots, L$.

$$G = \begin{bmatrix} I_s & 0 & s \\ 0 & A & L \\ 1_{(n-s-L) \times n} & n-s-L \end{bmatrix} \quad \quad A = \begin{bmatrix} 1_{2b+1} \\ 11111 \\ \vdots \\ 11111 \end{bmatrix}$$

Figure 3: Generator matrix used in Algorithm 1

$n, b$ settings. As a starting point, we state the result for $b = b_{\text{max}}$, i.e., a maximum Byzantine nodes setting. Proposition 1 implies that all data partitions need to be allocated to each worker, to guarantee perfect $b_{\text{max}}$–Byzantine tolerance. The proof of Proposition 1 is given in Appendix D.

**Proposition 1.** When $b = b_{\text{max}} = \lfloor n/2 \rfloor$, the system is perfect $b$–Byzantine tolerable if and only if the data allocation matrix is the all-one matrix, i.e., $G = 1_{n \times n}$. Thus, we have

$$r^*(n, b_{\text{max}}) = n.$$  

Now we provide deterministic codes that guarantee perfect $b$–Byzantine tolerance, when $0 < b < b_{\text{max}}$. The detailed code construction rule for generating matrix $G$ is given in Algorithm 1 and is depicted in Fig. 3. The example codes generated by this algorithm are given in Table I. In the theorem below, we provide the property of codes generated by Algorithm 1. The proof of this theorem is given in Appendix F.

**Theorem 3.** The deterministic code suggested in Algorithm 1 satisfies perfect $b$–Byzantine tolerance for $0 < b < b_{\text{max}}$, by utilizing the computational redundancy of

$$r(u) := \frac{n + (2b + 1)}{2} - \left( \left\lfloor \frac{n - (2b + 1)}{2(b+1)} \right\rfloor + \frac{1}{2} \right) \frac{n - (2b + 1)}{n}. \quad (13)$$
Table I: Examples of perfect $b$–Byzantine tolerable codes generated by Algorithm 1 when $n = 5$ or $n = 7$.

| $(n, b)$  | $r^{(u)}$ | $(5,1)$ | $(7,1)$ | $(7,2)$ |
|----------|-----------|---------|---------|---------|
| $n=5$    | 3.8       | 4.14    | 5.86    |         |
| $n=7$    |           |         |         |         |

\[ G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ \mathbb{1}_{3\times5} \end{bmatrix}, \quad G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ \mathbb{1}_{3\times7} \end{bmatrix}, \quad G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ \mathbb{1}_{5\times7} \end{bmatrix} \]

B. Analysis on the minimum required redundancy $r^*$

Here we provide some results on the minimum required redundancy $r^*$ for perfect Byzantine tolerance. First, we give a closed-form expression of $r^*$ when $n = 5$ or $n = 7$.

**Proposition 2.** The codes in Algorithm 1 has the minimum redundancy, i.e., $r^* = r^{(u)}$ holds for $r^{(u)}$ in (13), when $n = 5$ or $n = 7$.

Now, we provide upper and lower bounds on $r^*$ for general $n, b$ settings. Before stating the general bounds, we define a parameter $z$ which is useful for specifying the lower bound.

**Definition 3.** For given $n, b$, define

\[
\begin{align*}
    z := \max_{a_1, a_3, \ldots, a_{n-2}} \{(n - 1)a_1 + (n - 3)a_3 + \cdots + 2a_{n-2}\} \\
    \text{subject to} \quad a_1 + a_3 + \cdots + a_{n-2} \leq n, \\
    \sum_{t=1}^{\frac{n-1}{2}} \left( \begin{array}{c} 2t - 1 \\ i \end{array} \right) \left( \begin{array}{c} n - 2t + 1 \\ \frac{n-1}{2} - i \end{array} \right) \leq \left( \begin{array}{c} n \\ \frac{n-1}{2} \end{array} \right) \cdot \left( \begin{array}{c} n - 1 \\ 2 \end{array} \right), \\
    a_1 \leq \frac{n-1}{2} - b, \\
    a_1, a_3, \ldots, a_{n-2} \in \mathbb{Z}_0, \quad \text{where} \quad \mathbb{Z}_0 \quad \text{is the set of non-negative integers.}
\end{align*}
\]

Note that the parameter $z$ in (14) is the solution of an integer linear programming, which can be obtained from the simplex method [39]. In the theorem below, we provide upper and lower bounds on the minimum required redundancy $r^*$ for perfect $b$–Byzantine tolerance. The proof of this theorem is given in Appendix G.

**Theorem 4.** Consider designing the allocation matrix $G$ for perfect $b$–Byzantine tolerance. The
minimum required computational redundancy $r^*$ is bounded as

$$r^{(l)} \leq r^* \leq r^{(u)},$$

where the upper bound $r^{(u)}$ is in (13), and the lower bound is $r^{(l)} = n - \frac{z}{n}$ for $z$ in (14).

The upper and lower bounds are plotted in Fig. 4. As shown in the figure, both the upper and lower bounds increase linearly with $n$, when the portion of Byzantines $\alpha$ is fixed. Thus, the minimum required redundancy $r^*$ has the same behavior. We can observe that the bounds are tighter for smaller $n$ or larger $\alpha$.

V. EXPERIMENTS ON AMAZON EC2

Here we provide experimental results of the suggested coding schemes, tested on Amazon EC2. Considering a distributed learning setup with communication across multiple nodes, we used MPI4py [40], an open source message passing interface.

**Compared Schemes.** We compare the suggested coding schemes with the conventional uncoded scheme of SIGNSGD with MAJORITY VOTE. Similar to the simulation settings in the previous works [21], [22], we used the momentum counterpart SIGNUM instead of SIGNSGD for fast convergence, and used the learning rate of $\gamma = 0.001$ and the momentum term of $\eta = 0.9$. We simulated deterministic codes given in Algorithm 1 and Bernoulli codes suggested in Section III with connection probability of $p$. Thus, the probabilistic code have expected computational redundancy of $E[r] = np$.

**Byzantine Attack Model.** We consider the following two attack models used in related works [17], [22]: 1) the reverse attack where a Byzantine node flips the sign of the true gradient vector, and 2) the directional attack where a Byzantine node guides the model parameter in a certain
direction. Here, we set the direction as an all-one vector. For each experiment, $b$ Byzantine nodes are selected arbitrarily.

A. Experiments on Deep Neural Network Models

We trained a RESNET-18 model on CIFAR-10 dataset. Under this setting, the model dimension is $d = 11, 173, 962$, and the number of training/test samples is set to $n_{\text{train}} = 50000$ and $n_{\text{test}} = 10000$, respectively. We used mini-batch stochastic gradient descent; the batch size is set to $B = 120$ when $n = 5$, and set to $B = 126$ when $n = 9$. We used $g4dn.xlarge$ instances (having a GPU at each instance) for both workers and the master.

Fig. 5 illustrates the test accuracy performances of coded/uncoded schemes. Each curve represents an average over 3 independent train runs. The left column of the figure plots the performances when $n = 5, b = 1$, and the right column plots for the scenario of $n = 9, b = 2$. For each scenario, two types of plots are given: one having horizontal axis of training epoch, and the other with horizontal axis of training time. We plotted the case with no attack as a reference to an ideal scenario. As in Fig. 5a the system using the deterministic code achieves $15 - 20\%$ higher test accuracy at each epoch, compared to the uncoded case. Moreover, as in Fig. 5c the deterministic code achieves $10 - 15\%$ higher test accuracy than the uncoded scheme at each
training time. Thus, the suggested schemes achieve a given target test accuracy with less training time, compared to the uncoded scheme. We can also confirm that deterministic codes which guarantee zero estimation error follow the performance of an ideal scenario with no attack. In the case of the probabilistic code with a reasonable redundancy $E[r] = 2.5$, we can still enjoy 10% accuracy gap compared to the uncoded scenario at each epoch, and 5% accuracy gap at each training time, as in Figs. 5a and 5c. When $n = 9$ and $b = 2$, the test accuracy performances of various schemes are compared in Figs. 5b and 5d. Interestingly, probabilistic codes with the expected redundancy as small as $E[r] = 2.5$ nearly achieves the performance of the ideal scenario at each epoch. The suggested codes guarantee 10% accuracy gap compared to the uncoded case.

Now, what if the number of Byzantines is more than we expected? Are the suggested codes resilient to this mismatch scenario? We plotted the simulation results in Fig. 6. Again, each curve reflects an average over 3 independent runs. When $n = 5$, we used probabilistic/deterministic codes suitable for the scenario of having $\hat{b} = 1$ Byzantine, while the actual number of Byzantines is $b = 2$. When $n = 9$, we applied the suggested codes suitable for $\hat{b} = 2$ setting, while there are actually $b = 3$ Byzantines. In both plots, applying the suggested codes having a reasonable redundancy of $E[r] = 2.5$ or $E[r] = 3$ guarantees to enjoy $10-20\%$ accuracy gap at each training time, compared to the uncoded scheme. Thus, even under the Byzantine mismatch scenario, the suggested codes maintain a remarkable training time reduction to achieve a target test accuracy.
VI. CONCLUSIONS

In this paper, we proposed ELECTION CODING, a coding-theoretic framework that guarantees Byzantine tolerance of SIGNSGD WITH MAJORITY VOTE, a communication-efficient distributed learning algorithm. This framework tolerates arbitrary attacks corrupting the gradient computed in the training phase, by exploiting redundant gradient computations with appropriate allocation mapping between the individual workers and data partitions. Making use of majority-rule-based

3https://www.kaggle.com/c/amazon-employee-access-challenge
encoding as well as decoding functions, we suggested two types of codes that tolerate Byzantine attacks with a controlled amount of redundancy, namely, random Bernoulli codes and deterministic codes. The Byzantine tolerances of these codes are proved via mathematical analysis as well as through deep learning and logistic regression simulations on Amazon EC2.

**APPENDIX A**

**NOTATIONS USED FOR PROOFS**

We define notations used in proving main results. First, we denote the mapping between a message vector $m$ and a coded vector $c$ as $\phi(\cdot)$:

$$\phi(m) = c = [c_1, \cdots, c_n] = [E_1(m; G), \cdots, E_n(m; G)].$$

We also define the attack vector $\beta = [\beta_1, \beta_2, \cdots, \beta_n]$, where $\beta_j = 1$ if node $j$ is a Byzantine and $\beta_j = 0$ otherwise. The set of attack vectors with a given support $b$ is denoted as $B_b = \{\beta \in \{0, 1\}^n : \|\beta\|_0 = b\}$. For a given attack vector $\beta$, we define an attack function $f_\beta : c \mapsto y$ to represent the behavior of Byzantine nodes. According to the definition of $y_j$ in the main manuscript, the set of valid attack functions can be expressed as $F_\beta := \{f_\beta \in F : y_j = c_j \quad \forall j \in [n] \text{ with } \beta_j = 0\}$, where $F = \{f : \{0, 1\}^n \rightarrow \{0, 1\}^n\}$ is the set of all possible mappings. Moreover, the set of message vectors $m$ with weight $t$ is defined as

$$M_t := \{m \in \{0, 1\}^n : \|m\|_0 = t\}.$$  \hspace{1cm} (A.1)

Now we define several sets:

$$M^+ := \{m \in \{0, 1\}^n : \|m\|_0 > \left\lfloor \frac{n}{2} \right\rfloor\}, \quad M^- := \{m \in \{0, 1\}^n : \|m\|_0 \leq \left\lfloor \frac{n}{2} \right\rfloor\},$$

$$Y^+ := \{y \in \{0, 1\}^n : \|y\|_0 > \left\lfloor \frac{n}{2} \right\rfloor\}, \quad Y^- := \{y \in \{0, 1\}^n : \|y\|_0 \leq \left\lfloor \frac{n}{2} \right\rfloor\}.$$

Using these definitions, Fig. 8 provides a description on the mapping from $m$ to $\hat{\mu}$. Since decoder $D(\cdot)$ is a majority vote function, we have $\hat{\mu} = 1_{\{y \in Y^+\}}$. Moreover, we have $\mu = 1_{\{m \in M^+\}}$.

**APPENDIX B**

**PROOF OF THEOREM 1**

We begin with the following lemma, which can be obtained from the definition of $y_j$.  

...
Figure 8: Mapping from $m \in \{0,1\}^n$ to $\hat{\mu} \in \{0,1\}$. For an arbitrary attack vector $\beta \in B_b$ and an arbitrary attack function $f_{\beta} \in F_{\beta}$, we want the overall mapping satisfies $\hat{\mu} = 1$ for all $m \in M^-$ and $\hat{\mu} = 0$ for all $m \in M^+$.

**Lemma B.1.** Assume that there are $b$ Byzantine nodes, i.e., the attack vector satisfies $\beta \in B_b$. For a given vector $c$, the output $y = f_{\beta}(c)$ of an arbitrary attack function $f_{\beta} \in F_{\beta}$ satisfies $\|y \oplus c\|_0 \leq b$. In other words, $y$ and $c$ differ at most $b$ positions.

Now, define a randomly generated data allocation matrix $G$ to be *irregular* if it has at least one all-zero row, i.e., there is a node which does not have any data partitions. Then, the estimation error can be expressed as

$$P(\mu \neq \hat{\mu}) \leq P(G \text{ is regular}) \ P(\mu \neq \hat{\mu} \mid G \text{ is regular}) + P(G \text{ is irregular}) \quad \text{(B.1)}$$

First, we specify the second term on the right-hand side. Let $E_j$ be the event that node $j$ receives no data partitions. Then, $P(E_j) = (1 - p)^n$ holds. Moreover, from the independence of rows of matrix $G$, we obtain

$$P(G \text{ is irregular}) = 1 - P(\cap_{j=1}^n E_j^c) = 1 - \prod_{j=1}^n P(E_j^c) = 1 - (1 - (1 - p)^n)^n \approx 1 - (1 - n(1 - p)^n) = n(1 - p)^n = o(1) \quad \text{(B.2)}$$

as $n$ increases. Now we focus on the first term on the right-hand side of (B.1). We first develop an upper bound on $P(\mu \neq \hat{\mu} \mid G \text{ is regular})$ as follows. Given that $G$ is regular, we write

$$P(\mu \neq \hat{\mu}) = \sum_{m \in \{0,1\}^n} P(m)P(\hat{\mu} \neq \mu \mid m) = \sum_{m \in M^-} P(m)P(\hat{\mu} \neq \mu \mid m) + \sum_{m \in M^+} P(m)P(\hat{\mu} \neq \mu \mid m)$$

$$= \frac{1}{2^n} \left\{ \sum_{m \in M^-} P(\hat{\mu} \neq \mu \mid m) + \sum_{m \in M^+} P(\hat{\mu} \neq \mu \mid m) \right\} = \frac{1}{2^{n-1}} \sum_{m \in M^-} P(\hat{\mu} \neq \mu \mid m) \quad \text{(B.3)}$$

where the second last equality is from the assumption that each message vector is equally likely while the last equality holds since the analysis below for $m \in M^-$ can be similarly applied to
the case of $m \in M^+$. Next, obtain an upper bound on $P(\hat{\mu} \neq \mu|m)$. For an arbitrary $m \in M^-$,

$$P(\hat{\mu} \neq \mu|m) = P(\hat{\mu} = 1|m) = P(\|f_\beta(\phi(m))\|_0 > [n/2] | m)$$

$$\leq P(\|\phi(m)\|_0 > [n/2] - b | m) = P(\sum_{j=1}^{n} 1_{\{c_j = 1\}} > [n/2] - b | m)$$ (B.4)

where the inequality is from Lemma B.1. Consider an arbitrary $m \in M^-$ which has weight $w$, i.e., $\|m\|_0 = w$. Now define

$$q_w := P(c_j = 1 | m),$$ (B.5)

which can be obtained as follows. Recall that node $j$ obtains data partition $i$ if $G_{ji} = 1$ holds. Thus, the number of data partitions observed from node $j$ is $|P_j| = \|G(j,:)\|_0$ where $P_j = \{i \in [n]: G_{ji} = 1\}$. Observing these $|P_j|$ partitions, node $j$ generates $c_j = E_j(m; G) = \text{maj}(\{m_i\}_{i \in P_j})$. Thus, for a given message vector $m$, the number of data partitions yielding a signed gradient of 1 and also observed by node $j$ can be expressed as $\sum_{i \in P_j} 1_{\{m_i = 1\}} = m^TG(j,:)$. Thus, the event $c_j = 1$ occurs if the majority of $\{m_i\}_{i \in P_j}$ is one, i.e., $\|G(j,:)\|_0 \leq 2m^TG(j,:)$ holds. In a mathematical form, we can express that the event $c_j = 1$ happens when both $\|G(j,:)\|_0 = v$ and $m^TG(j,:) \geq [v/2]$ hold for some $v = 1, 2, \cdots, 2w$. Thus, $q_w$ in (B.5) can be expressed as

$$q_w = \sum_{v=1}^{2w} P(c_j = 1 | m, \|G(j,:)\|_0 = v) P(\|G(j,:)\|_0 = v)$$ (B.6)

Here, we denote the first probability term as

$$q_v := P(c_j = 1 | m, \|G(j,:)\|_0 = v) = P(m^TG(j,:) \geq [v/2] | m, \|G(j,:)\|_0 = v)$$

which can be simply calculated as $q_v = \sum_{i=\lceil v/2 \rceil}^{\min\{w,v\}} \binom{w}{i} \binom{n-w}{v-i} / \binom{n}{v}$. Combining the equation for $q_v$ with $P(\|G(j,:)\|_0 = v) = \binom{n}{v} p^v(1-p)^{n-v}$, (B.6) reduces to

$$q_w = \sum_{v=1}^{2w} \sum_{i=\lceil v/2 \rceil}^{\min\{w,v\}} \binom{w}{i} \binom{n-w}{v-i} p^v(1-p)^{n-v}. \quad \text{(B.7)}$$

Now we obtain the following bound on $P(\mu \neq \hat{\mu})$. When the message vector has weight $w$, i.e.,

4When a tie occurs in the majority vote function, we count it as an error event. In other words, when $\sum_{i \in P_j} 1_{\{m_i = 1\}} = \sum_{i \in P_j} 1_{\{m_i = 0\}}$, we assume that node $j$ generates $c_j = 1$. 
∥m∥₀ = w, we have

\[ P(\sum_{j=1}^{n} 1_{c_j=1} > \lfloor n/2 \rfloor - b \mid m) = \sum_{s=\lfloor n/2 \rfloor - b+1}^{n} \binom{n}{s} q_w^s (1 - q_w)^{n-s} \]  \hspace{1cm} (B.8)

using the definition of \( q_w \) in (B.5). Thus, combining (B.3), (B.4), (B.7) and (B.8), we have

\[ \delta' := P(\mu \neq \hat{\mu} \mid G \text{ is regular}) = \frac{1}{2^{n-1}} \sum_{m \in M} P(\hat{\mu} \neq \mu \mid m) \leq \frac{1}{2^{n-1}} \sum_{m \in M} P(\sum_{j=1}^{n} 1_{c_j=1} > \lfloor n/2 \rfloor - b \mid m) \]  \hspace{1cm} (B.9)

\[ = \frac{1}{2^{n-1}} \sum_{w=1}^{\lfloor k/2 \rfloor} \binom{n}{w} \sum_{s=\lfloor n/2 \rfloor - b+1}^{n} \binom{n}{s} q_w^s (1 - q_w)^{n-s} \]  \hspace{1cm} (B.10)

where the last equality is from the fact that there exist \( \binom{n}{w} \) message vectors \( m \) having weight \( \|m\|_0 = \omega \). Combining (B.1), (B.2) and (B.10) completes the proof.

APPENDIX C

PROOF OF THEOREM 2

Let an attack vector \( \beta \) and an attack function \( f_\beta(\cdot) \) given. Consider an arbitrary \( m \in M^+ \). From the definitions of \( \mu \) and \( \hat{\mu} \), we have \( \mu = \hat{\mu} \) iff \( f_\beta(\phi(m)) \in Y^+ \). Similarly, for an arbitrary \( m \in M^- \), we have \( \mu = \hat{\mu} \) iff \( f_\beta(\phi(m)) \in Y^- \). Thus, from the definitions of \( Y^+ \) and \( Y^- \), the sufficient and necessary condition for \( b \)-Byzantine tolerance can be expressed as follows.

**Proposition 3.** The perfect \( b \)-Byzantine tolerance condition in Definition 7 is equivalent to the following: \( \forall \beta \in B_b, \forall f_\beta \in F_\beta, \)

\[ \begin{cases} 
\|f_\beta(\phi(m))\|_0 > \lfloor n/2 \rfloor, & \forall m \in M^+ \\
\|f_\beta(\phi(m))\|_0 \leq \lfloor n/2 \rfloor, & \forall m \in M^-
\end{cases} \]  \hspace{1cm} (C.1)

The condition stated in Proposition 3 can be further simplified as follows.

**Proposition 4.** The perfect \( b \)-Byzantine tolerance condition in Proposition 3 is equivalent to

\[ \begin{cases} 
\|\phi(m)\|_0 > \lfloor n/2 \rfloor + b, & \forall m \in M^+ \\
\|\phi(m)\|_0 \leq \lfloor n/2 \rfloor - b, & \forall m \in M^-
\end{cases} \]  \hspace{1cm} (C.2)
Proof. Consider arbitrary $m \in M^-$. We want to prove that
\[
\forall \beta \in B_0, \forall f_\beta \in F_\beta, \quad \|f_\beta(\phi(m))\|_0 \leq \left\lfloor \frac{n}{2} \right\rfloor \tag{C.3}
\]
is equivalent to
\[
\|\phi(m)\|_0 \leq \left\lfloor \frac{n}{2} \right\rfloor - b. \tag{C.4}
\]
First, we show that (C.4) implies (C.3). According to Proposition B.1 $\|f_\beta(\phi(m))\oplus \phi(m)\|_0 \leq b$ holds for arbitrary $\beta \in B_0$ and arbitrary $f_\beta \in F_\beta$. Thus,
\[
\|f_\beta(\phi(m))\|_0 \leq \|f_\beta(\phi(m))\oplus \phi(m)\|_0 + \|\phi(m)\|_0 \leq b + \left(\left\lfloor \frac{n}{2} \right\rfloor - b \right) = \left\lfloor \frac{n}{2} \right\rfloor
\]
holds for $\forall \beta \in B_0, \forall f_\beta \in F_\beta$, which completes the proof. Now, we prove that (C.3) implies (C.4), by contra-position. Suppose $\|\phi(m)\|_0 > \left\lfloor \frac{n}{2} \right\rfloor - b$. We divide the proof into two cases. The first case is when $\|\phi(m)\|_0 > n - b$. In this case, we arbitrary choose $\beta^\star \in B_0$ and select the identity mapping $f_{\beta^\star}^\star : c \mapsto y$ such that $y_j = c_j$ for all $j \in [n]$. Then, $\|f_{\beta^\star}^\star(\phi(m))\|_0 = \|\phi(m)\|_0 > n - b \geq n - \lfloor n/2 \rfloor \geq \lfloor n/2 \rfloor$. Thus, we can state that
\[
\exists \beta^\star \in B_0, \exists f_{\beta^\star}^\star \in F_{\beta^\star} \text{ such that } \|f_{\beta^\star}^\star(\phi(m))\|_0 \leq \left\lfloor \frac{n}{2} \right\rfloor
\]
when $\|\phi(m)\|_0 > n - b$, which completes the proof for the first case. Now consider the second case where $\lfloor n/2 \rfloor - b < \|\phi(m)\|_0 \leq n - b$. To begin, denote $\phi(m) = c = [c_1, c_2, \ldots, c_n]$. Let $S = \{i \in [n] : c_i = 0\}$, and select $\beta^\star \in B_0$ which satisfies $\{i \in [n] : \beta^\star_i = 1\} \subseteq S$. Now define $f_{\beta^\star}^\star(\cdot)$ as $f_{\beta^\star}^\star(\phi(m)) = \phi(m) \oplus \beta^\star$. Then, we have
\[
\|f_{\beta^\star}^\star(\phi(m))\|_0 = \|\phi(m)\|_0 + \|\beta^\star\|_0 > \left\lfloor \frac{n}{2} \right\rfloor - b + b = \left\lfloor \frac{n}{2} \right\rfloor.
\]
Thus, the proof for the second case is completed, and this completes the statement of (C.2) for arbitrary $m \in M^-$. Similarly, we can show that
\[
\forall \beta \in B_0, \forall f_\beta \in F_\beta, \quad \|f_\beta(\phi(m))\|_0 > \left\lfloor \frac{n}{2} \right\rfloor
\]
is equivalent to $\|\phi(m)\|_0 > \left\lfloor \frac{n}{2} \right\rfloor + b$ for arbitrary $m \in M^+$. This completes the proof. \qed

\footnote{We can always find such $\beta^\star$ since $|S| \geq b$ due to the setting of $\|\phi(m)\|_0 \leq n - b$.}
Now, we further reduce the condition in Proposition 4 as follows.

**Proposition 5.** The perfect $b$--Byzantine tolerance condition in Proposition 4 is equivalent to

$$\|\phi(m)\|_0 \leq \left\lfloor \frac{n}{2} \right\rfloor - b, \quad \forall m \in M^-$$  \hspace{1cm} (C.5)

**Proof.** All we need to prove is that (C.5) implies (C.2). Assume that the mapping $\phi$ satisfies (C.5). Consider an arbitrary $m' \in M^+$ and denote $m' = [m'_1, m'_2, \cdots, m'_n]$. Define $m = [m_1, m_2, \cdots, m_n]$ such that $m'_i \oplus m_i = 1$ for all $i \in [n]$. Then, we have $m \in M^-$ from the definitions of $M^+$ and $M^-$. Now we denote $\phi(m) = c = [c_1, c_2, \cdots, c_n]$ and $\phi(m') = c' = [c'_1, c'_2, \cdots, c'_n]$. Then, $c_j \oplus c'_j = 1$ holds for all $j \in [n]$ since $E_j(\cdot)$ is a majority vote function. In other words, $\|\phi(m)\|_0 + \|\phi(m')\|_0 = n$ holds. Thus, if a given mapping $\phi$ satisfies $\|\phi(m)\|_0 \leq \left\lfloor \frac{n}{2} \right\rfloor - b$ for all $m \in M^-$, then $\|\phi(m')\|_0 \geq n - (\left\lfloor \frac{n}{2} \right\rfloor - b) = \left\lfloor \frac{n}{2} \right\rfloor + b = \left\lfloor \frac{n}{2} \right\rfloor + b$ holds for all $m \in M^+$, which completes the proof. \hfill \Box

In order to prove Theorem 2, all that remains is to prove that (C.5) reduces to

$$\sum_{v=1}^{\left\lfloor \frac{n}{2} \right\rfloor} |J_v(m)| \leq \left\lfloor \frac{n}{2} \right\rfloor - b \quad \forall m \in M_{\left\lfloor \frac{n}{2} \right\rfloor}.$$  \hspace{1cm} (C.6)

Recall that $\phi(m) = c = [c_1, c_2, \cdots, c_n]$ where $c_j = \text{maj}(\{m_i\}_{i \in P_j})$ and $P_j = \{i \in [n] : G_{ji} = 1\}$. Moreover, we assumed that $|P_j| = \|G(j, :)|_0$ is an odd number. Thus, $c_j = 1$ if $\|G(j, :)|_0 + 1 \leq 2m^T G(j, :)$, and the set $[n] = \{1, 2, \cdots, n\}$ can be partitioned as $[n] = S_1 \cup S_2 \cup \cdots \cup S_{\left\lfloor \frac{n}{2} \right\rfloor + 1}$ where $S_v := \{j \in [n] : \|G(j, :)|_0 = 2v - 1\}$. Therefore, for a given $m \in M^-$, we have

$$\|\phi(m)\|_0 = \sum_{j=1}^{n} c_j = \sum_{v=1}^{\left\lfloor \frac{n}{2} \right\rfloor + 1} |\{j \in S_v : c_j = 1\}|$$

$$= \sum_{v=1}^{\left\lfloor \frac{n}{2} \right\rfloor + 1} \left| \left\{ j \in S_v : m^T G(j, :) \geq \frac{\|G(j, :)\|_0 + 1}{2} + 1 = v \right\} \right| = \sum_{v=1}^{\left\lfloor \frac{n}{2} \right\rfloor + 1} |J_v(m)|.$$  \hspace{1cm}

Note that $J_v(m)$ for $v = \left\lfloor \frac{n}{2} \right\rfloor + 1$ reduces to

$$J_{\left\lfloor \frac{n}{2} \right\rfloor + 1}(m) = \{j \in [n] : \|G(j, :)\|_0 = 2(\left\lfloor \frac{n}{2} \right\rfloor - 1) + 1, m^T G(j, :) \geq \left\lfloor \frac{n}{2} \right\rfloor + 1 \} = \emptyset$$

---

6Recall that $c_j = E_j(\{m_i\}_{i \in P_j}) = \text{maj}(\{m_i\}_{i \in P_j})$ and $c'_j = \text{maj}(\{m'_i\}_{i \in P_j})$. Thus, $m'_i \oplus m_i = 1$ for all $i \in [n]$ implies that $c_j \oplus c'_j = 1$ holds for all $j \in [n]$.---
since \( m \in M^- \). Thus, combining the two equations above, we obtain the following.

**Proposition 6.** The perfect \( b \)-Byzantine tolerance condition in Proposition 5 is equivalent to

\[
\sum_{v=1}^{\lfloor n/2 \rfloor} |J_v(m)| \leq \left\lfloor \frac{n}{2} \right\rfloor - b \quad \forall m \in M^-,
\]

or equivalently,

\[
\sum_{v=1}^{\lfloor n/2 \rfloor} |J_v(m)| \leq \left\lfloor \frac{n}{2} \right\rfloor - b \quad \forall m \in M_t, \quad \forall t = 0, 1, \ldots, \lfloor n/2 \rfloor.
\]  

(C.7)

Now, we show that (C.7) is equivalent to (C.6). We can easily check that the former implies the latter, which is directly proven from the statements. Thus, all we need to prove is that (C.6) implies (C.7). First, when \( t = 0 \), note that \( |J_v(m)| = 0 \) for \( \forall m \in M_0, \forall v \in \{1, 2, \ldots, \lfloor n/2 \rfloor\} \), which implies that (C.7) holds trivially. Thus, in the rest of the proof, we assume that \( t > 0 \).

Consider an arbitrary \( t \in \{1, 2, \ldots, \lfloor n/2 \rfloor\} \) and an arbitrary \( m \in M_t \). Denote \( m = e_{i_1} + e_{i_2} + \cdots + e_{i_t} \) where \( e_1 = [1, 0, \ldots, 0] \), \( e_2 = [0, 1, 0, \ldots, 0] \), and \( e_n = [0, \ldots, 0, 1] \). Moreover, consider an arbitrary \( m' \in M_{\lfloor n/2 \rfloor} \) which satisfies \( m'_i = 1 \) for \( i = i_1, i_2, \ldots, i_t \). Denote \( m' = e_{i_1} + \cdots + e_{i_t} + e_{j_1} + \cdots + e_{j_{\lfloor n/2 \rfloor}} \). Then, \( (m' - m)^T G(j, \cdot) \geq 0 \) holds for all \( j \in [n] \), which implies \( J_v(m) \subseteq J_v(m') \) for all \( v = 1, 2, \ldots, \lfloor n/2 \rfloor \). Thus, we have \( |J_v(m)| \leq |J_v(m')| \) for all \( v \in \{1, 2, \ldots, \lfloor n/2 \rfloor\} \), which implies \( \sum_{v=1}^{\lfloor n/2 \rfloor} |J_v(m)| \leq \sum_{v=1}^{\lfloor n/2 \rfloor} |J_v(m')| \). Since this holds for arbitrary \( m' \in M_{\lfloor n/2 \rfloor} \), \( m \in M_t \), and \( t \in \{1, 2, \ldots, \lfloor n/2 \rfloor\} \), we can conclude that (C.6) implies (C.7). All in all, (C.7) is equivalent to (C.6). Combining this with Propositions 3, 4, 5 and 6 completes the proof of Theorem 2.

**APPENDIX D**

**PROOF OF PROPOSITION 1**

First, we prove that a system using \( G = 1_{n \times n} \) can tolerate \( b = \lfloor n/2 \rfloor \) Byzantine nodes. Note that since \( G = 1_{n \times n} \), we have \( P_j = [n] \) for all \( j \in [n] \). Consider arbitrary \( m \in \{0, 1\}^n \) and \( \beta \in B_b \) are given. Denote \( B = \{ j \in [n] : \beta_j = 1 \} \). where \( |[n] \setminus B| = n - \lfloor n/2 \rfloor = \lfloor n/2 \rfloor + 1 \). Then, \( y_j = c_j = \text{maj}(\{m_j\}_{j \in P_j}) = \text{maj}(\{m_j\}_{j \in [n]}) = \mu \) holds for all \( j \in [n] \setminus B \), i.e., each non-Byzantine node \( j \) outputs \( y_j = \mu \). Thus, \( \hat{\mu} = \text{maj}(\{y_j\}_{j \in [n]}) = \mu \) holds regardless of the attack function \( f_\beta \in F_\beta \) of Byzantine nodes. Therefore, by the definition of \( b \)-Byzantine tolerance conditions.
in Definition 1 (in the main manuscript), we can conclude that \( b = \lfloor n/2 \rfloor \) Byzantine nodes are tolerable when we use the allocation matrix \( G = 1_{n \times n} \). This completes the first part of the proof.

Second, we prove that if a system uses \( G \neq 1_{n \times n} \), it cannot tolerate \( b = \lfloor n/2 \rfloor \) Byzantine nodes. Note that since \( G \neq 1_{n \times n} \), there exists a node \( j_0 \in [n] \) such that \( P_{j_0} \neq [n] \). Now we aim at finding \( m \in \{0, 1\}^n, \beta \in B_b, f_{\beta} \in F_{\beta} \) triple which satisfies \( \hat{\mu} \neq \mu \). Depending on the cardinality of \( P_j \), the remaining proofs are different.

**Case I:** Consider the scenario where \( |P_{j_0}| \leq \lfloor n/2 \rfloor \) holds. First, set \( m \in \{0, 1\}^n \) as \( m_i = 1_{\{i \in P_{j_0}\}} \) for \( i \in [n] \). Then, \( \mu = \text{maj}(\{m_i\}_{i \in [n]}) = 0 \) holds. Now set an arbitrary \( \beta \in B_b \) such that \( \beta_{j_0} = 0 \) holds. Moreover, set \( f_\beta \in F_{\beta} \) such that the elements of \( y = f_{\beta}(c) \) is \( y_j = 1_{\{j = 1\}} + c_j \cdot 1_{\{j = 0\}} \) Then, \( y_{j_0} = c_{j_0} = \text{maj}(\{m_i\}_{i \in P_{j_0}}) = 1 \), and

\[
|\{j \in [n] : y_j = 1\}| \geq |\{j_0\}| + |\{j \in [n] : \beta_j = 1\}| = 1 + b = \lfloor n/2 \rfloor + 1 \tag{D.1}
\]

hold. Thus, \( \hat{\mu} = \text{maj}(\{y_j\}_{j \in [n]}) = 1 \neq \mu \) holds.

**Case II:** Consider the scenario with \( |P_j| > \lfloor n/2 \rfloor \). First, set arbitrary \( Q_{j_0} \subseteq P_{j_0} \) such that \( |Q_{j_0}| = \lfloor n/2 \rfloor \). Moreover, set \( m \in \{0, 1\}^n \) as \( m_i = 1_{\{i \in Q_{j_0}\}} \) for \( i \in [n] \). Then, \( \mu = \text{maj}(\{m_i\}_{i \in [n]}) = 0 \) holds. Now set \( \beta \in B_b \) and \( f_\beta \in F_{\beta} \) as in Case I. Then, \( y_{j_0} = c_{j_0} = \text{maj}(\{m_i\}_{i \in P_{j_0}}) = 1_{\{|P_{j_0} \setminus Q_{j_0}| < |Q_{j_0}|\}} = 1 \) holds since \( |Q_{j_0}| = \lfloor n/2 \rfloor \) and \( |P_{j_0}| \leq n - 2 \). Here, \( |P_{j_0}| \leq k - 2 \) holds from the following three facts: 1) \( |P_{j_0}| \) and \( n \) are odd values, 2) \( P_{j_0} \neq [n] \), and 3) \( P_{j_0} \subseteq [n] \). Thus, (D.1) holds, which implies \( \hat{\mu} = \text{maj}(\{y_j\}_{j \in [n]}) = 1 \neq \mu \).

All in all, in both cases, we confirm that

\[
\exists m \in \{0, 1\}^n, \exists \beta \in B_b, \exists f_\beta \in F_{\beta} \text{ such that } \hat{\mu} \neq \mu
\]

when \( G \neq 1_{n \times n} \). This proves that a system using \( G \neq 1_{n \times n} \) cannot tolerate \( b = \lfloor n/2 \rfloor \) Byzantines.

**APPENDIX E**

**PROOF OF PROPOSITION 2**

We begin with the case \( b = 0 \). First, it is obvious that \( \|G\|_0 \geq n \) since at least one data partition is assigned to each node. Next, when \( G = I_n \), we have

\[
\sum_{v=1}^{\lfloor n/2 \rfloor} |J_v(m)| = |J_1(m)| = \|m\|_0 = \lfloor n/2 \rfloor
\]
for all \( m \in M_{\lfloor n/2 \rfloor} \). Thus, from Theorem 2, using the allocation matrix of \( G = I_n \) can tolerate \( b = 0 \) Byzantine node, i.e., the master can successfully estimate \( \mu \) in the system with no Byzantine nodes. In the case of \( b = b_{\text{max}} = \lfloor n/2 \rfloor \), the result of Corollary 3 directly proves Corollary 4.

The rest of the proof deals with the case of \( 0 < b < b_{\text{max}} \). First we define \( s_v = |\{j \in [n] : \|G(j,:)\|_0 = v\}| \), which represent the number of nodes having \( v \) data partitions. Trivially, we have \( \sum_{v=1,v: \text{odd}}^n s_v = n \). Now we begin the proof for the case of \((n, b) = (5, 1)\). According to Theorem 2, the perfect \( b \)–Byzantine tolerance condition is equivalent to the following:

\[
|J_1(m)| + |J_2(m)| \leq 1, \quad \forall m \in M_2. \tag{E.1}
\]

We have the following lemma on the condition of \( s_1 \) in order to satisfy (E.1).

**Lemma E.1.** Consider the scenario \((n, b) = (5, 1)\). If the perfect \( b \)–Byzantine tolerance condition in (E.1) holds, then we have \( s_1 \leq 1 \).

**Proof.** Suppose \( s_1 \geq 2 \). Then there exist \( j_1, j_2 \in [n] \) such that \( G(j_1,:) = e_{i_1} \) and \( G(j_2,:) = e_{i_2} \) for some \( i_1, i_2 \in [n] \). If \( i_1 = i_2 \), consider a message vector \( m \in M_2 \) with \( m_{i_1} = 1 \). Then, \( j_1, j_2 \in J_1(m) \). This implies \( |J_1(m)| = 2 \), which does not satisfy (E.1). If \( i_1 \neq i_2 \), consider a message vector \( m = e_{i_1} + e_{i_2} \) which satisfies \( m \in M_2 \). Then, \( j_1, j_2 \in J_1(m) \). This implies \( |J_1(m)| = 2 \), which does not satisfy (E.1). Thus, \( s_1 \geq 2 \) implies that the system is not perfect \( b \)–Byzantine tolerance, which completes the proof. \( \square \)

Now we have the following lemma on the condition of \( s_3 \) in order to satisfy (E.1).

**Lemma E.2.** Consider the scenario of \((n, b) = (5, 1)\). If the perfect \( b \)–Byzantine tolerance condition in (E.1) holds, then we have \( s_3 \leq 2 \).

**Proof.** Suppose \( s_3 \geq 3 \). Then there exists \( j_1, j_2, j_3 \in [n] \) such that

\[
\|G(j_1,:)\|_0 = \|G(j_2,:)\|_0 = \|G(j_3,:)\|_0 = 3. \tag{E.2}
\]

Define \( G' = G([j_1, j_2, j_3], :) \), a matrix consisting of \( j_1, j_2, j_3 \)-th row vectors extracted from \( G \). Thus, \( G' \in \{0, 1\}^{3 \times 5} \) holds. For \( i \in [5] \), define \( a_i = \|G'(i,:)\|_0 \), the support of the \( i \)-th column.
of $G'$. Then, from (E.2), we obviously have
\[
\sum_{i=1}^{5} a_i = 9,
\] (E.3)
and from the definition of $a_i$, we have $0 \leq a_i \leq 3$ for all $i \in [5]$.

We proceed the rest of the proof for three different cases which cover all the possible scenarios.

Case I: $\exists a_{i_1} = 3, \exists a_{i_2} \geq 2$

Consider the case of having distinct $i_1, i_2 \in [5]$ such that $a_{i_1} = 3$ and $a_{i_2} \geq 2$. Then, $G'$ consists of a sub-matrix of $M = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$. Without a loss of generality, let $G([j_1, j_2], [i_1, i_2]) = M$. Then, for a given message vector $m = e_{i_1} + e_{i_2}$, we have $j_1, j_2 \in J_2(m)$. Thus, this case does not satisfy (E.1).

Case II: $\exists a_{i_1} = 3$, and $a_{i_2} \leq 1$ for all $i_2 \neq i_1$

Consider the case where $i_1 \in [5]$ with $a_{i_1} = 3$, while all other indexes $i_2 \in [5]$ satisfy $a_{i_2} \leq 1$. In such a case, we have $\sum_{i=1}^{5} a_i = a_{i_1} + \sum_{i_2 \neq i_1} a_{i_2} \leq 7$, which contradicts (E.3).

Case III: $a_i \leq 2$ for all $i \in [5]$

Combining with (E.3), we have at least four indexes $\{i_p\}_{p=1}^{4} \in [5]$ such that $a_{i_p} = 2$. Note that since $G'$ has three rows, there are $\binom{3}{2} = 3$ distinct columns $G'(:, i_p)$ with weight $a_{i_p} = 2$. Thus, there exist distinct indexes $i_p, i_q \in [5]$ such that $G'(\cdot, i_p) = G'(\cdot, i_q)$ holds, which implies that $G'$ consists of a sub-matrix of $M = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$. Without a loss of generality, let $G([j_1, j_2], [i_p, i_q]) = M$. Then, for a given message vector $m = e_{i_p} + e_{i_q}$, we have $j_1, j_2 \in J_2(m)$. Thus, this case cannot satisfy (E.1). All in all, when we suppose $s_3 \geq 3$, all possible scenarios cannot satisfy (E.1). Thus, the condition $s_3 \leq 2$ is necessary for satisfying (E.1).

Finally, we show the following lemma stating the necessary condition on $s_1 + s_3$.

Lemma E.3. Consider the scenario $(n, b) = (5, 1)$. If the perfect $b$–Byzantine tolerance condition in (E.1) holds, then we have $s_1 + s_3 \leq 2$.

Proof. When $s_1 = 0$, Lemma [E.2] directly proves this Lemma. Since $s_1 \leq 1$ from Lemma [E.1], all we need to consider is the case $s_1 = 1$. Given $s_1 = 1$, suppose $s_3 \geq 2$. Then, there exist distinct $j_1, j_2, j_3 \in [5]$ such that $\|G(j_1, :)\|_0 = 1$ and $\|G(j_2, :)\|_0 = \|G(j_3, :)\|_0 = 3$. Note that from the pigeonhole principle, there exists at least one $i \in [5]$ which satisfies $G(j_2, i) = G(j_3, i) = 1$. In
other words, $|S| \geq 1$ holds for $S = \{ i \in [5] : G_{j_2,i} = G_{j_3,i} = 1 \}$.

We proceed with the rest of the proof for two different cases which would cover all possible scenarios.

**Case I** (when $|S| \geq 2$): Let $i_1, i_2 \in S$. Then, $G([j_2, j_3], [i_1, i_2]) = 1_{2 \times 2}$ holds. Then, for a given vector $m = e_{i_1} + e_{i_2}$, we have $j_2, j_3 \in J_2(m)$. Thus, this case does not satisfy (E.1).

**Case II** (when $|S| = 1$): Note that

$$\sum_{i=1}^{5} 1\{ G(j_2,i)=1 \text{ or } G(j_3,i)=1 \} = \sum_{i=1}^{5} 1\{ G(j_2,i)=1 \} + 1\{ G(j_3,i)=1 \} - 1\{ G(j_2,i)=1 \text{ and } G(j_3,i)=1 \}$$

$$= \| G(j_2,:) \|_0 + \| G(j_3,:) \|_0 - |S| = 3 + 3 - 1 = 5$$

holds. This implies that for arbitrary $i \in [n]$, either $G(j_2, i) = 1$ or $G(j_3, i) = 1$ holds.

Recall that $\| G(j_1,:) \|_0 = 1$ holds. Thus, $G(j_1, i_1) = 1$ holds for some $i_1 \in [5]$. Moreover, either $G(j_2, i_1) = 1$ or $G(j_3, i_1) = 1$ holds. Without a loss of generality, assume that $G(j_2, i_1) = 1$ holds. Since $\| G(j_2,:) \|_0 = 3$, there exists $i_2 \in [5]$ such that $G(j_2, i_2) = 1$. Therefore, we have $G([j_1, j_2], [i_1, i_2]) = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$. Thus, for a given message vector $m = e_{i_1} + e_{i_2}$, we have $j_1 \in J_1(m)$ and $j_2 \in J_2(m)$. Thus, this case does not satisfy (E.1). All in all, given $s_1 = 1$, we require $s_3 \leq 1$ to satisfy (E.1). This completes the proof.

From Lemmas E.1, E.2 and E.3, we have three candidates: $(s_1, s_3) = (0, 2), (1, 1), \text{ and } (2, 0)$. The computational burden of such candidates are expressed as $\| G \|_0 = s_1 + 3s_3 + 5(5 - s_1 - s_3) = 25 - 4s_1 - 2s_3$. The choice which minimizes the computational load $\| G \|_0$ is $(s_1, s_3) = (1, 1)$. This has computational redundancy of $\| G \|_0/n = 19/5 = 3.8$, which can be achieved by using a matrix in Table II with $(s_1, s_3) = (1, 1)$. This completes the proof for $(n, b) = (5, 1)$. Using a similar analysis, we can obtain results for $n = 7$.

**APPENDIX F**

**PROOF OF THEOREM 3**

**Proof.** Recall that according to Theorem 2 and the definition of $M_t$ in (A.1), the system using the allocation matrix $G$ is perfect $b$—Byzantine tolerable if and only if

$$\sum_{v=1}^{(n-1)/2} |J_v(m)| \leq \frac{n - 1}{2} - b \quad (F.1)$$
holds for arbitrary message vector \( m \in M_{n-1} \), where

\[
J_v(m) = \{ j \in [n] : m^T G(j,:) \geq v, \| G(j,:) \|_0 = 2v - 1 \}.
\]

Note that we have

\[
\| G(j,:) \|_0 = \begin{cases} 
1, & 1 \leq j \leq s \\
2b + 1, & s + 1 \leq j \leq s + L \\
n, & s + L + 1 \leq j \leq n.
\end{cases}
\] (F.2)

from Fig. 3. Thus, the condition in (F.1) reduces to

\[
|J_1(m)| + |J_{b+1}(m)| \leq s.
\] (F.3)

Now all that remains is to show that (F.3) holds for arbitrary message vector \( m \in M_{n-1} \).

Consider a message vector \( m \in M_{n-1} \) denoted as \( m = [m_1, m_2, \cdots, m_n] \). Here, we note that

\[
J_1(m) \subseteq \{1, 2, \cdots, s\}, \quad J_{b+1}(m) \subseteq \{s + 1, s + 2, \cdots, s + L\}
\] (F.4)

hold from Fig. 3. Define

\[
v(m) := |\{i \in \{s + 1, s + 2, \cdots, n\} : m_i = 1\}|,
\] (F.5)

which is the number of 1’s in the last \((n - s)\) coordinates of message vector \( m \). Since \( m \in M_{n-1} \), we have

\[
|\{i \in \{1, 2, \cdots, s\} : m_i = 1\}| = \frac{n - 1}{2} - v(m).
\] (F.6)

Note that since \( G(1:s,:) = [I_s \mid 0_{s \times (n-s)}] \), we have

\[
m^T G(j,:) = 1_{\{m_j = 1\}}, \quad \| G(j,:) \|_0 = 1, \quad \forall j \in [s].
\] (F.7)

Combining (10), (F.4), (F.6), and (F.7), we have \( |J_1(m)| = \frac{n - 1}{2} - v(m) \). Now, in order to obtain (F.3), all we need to prove is to show

\[
|J_{b+1}(m)| \leq s - \left( \frac{n - 1}{2} - v(m) \right) \overset{(a)}{=} v(m) - b
\] (F.8)
where (a) is from the definition of $s$ in Algorithm 1. We alternatively prove that

$$\text{if } \left| J_{b+1}(m) \right| \geq q \text{ for some } q \in \{0, 1, \ldots, L\}, \text{ then } v(m) \geq b + q. \tag{F.9}$$

Using the definition $M(q) := \{m \in M_{\geq q} : \left| J_{b+1}(m) \right| \geq q\}$, the statement in (F.9) is proved as follows: for arbitrary $q \in \{0, 1, \ldots, L\}$, we first find the minimum $v(m)$ among $m \in M(q)$, i.e., we obtain a closed-form expression for $v_q^* := \min_{m \in M(q)} v(m)$. \tag{F.10}

Second, we show that $v_q^* \geq b + q$ holds for all $q \in \{0, 1, \ldots, L\}$, which completes the proof.

The expression for $v_q^*$ can be obtained as follows. Fig. 9 supports the explanation. First, define

$$M_{\text{gather}}(q) := \{m \in M(q) : \text{if } j, j + 2 \in J_{b+1}(m) \text{, then } j + 1 \in J_{b+1}(m)\}, \tag{F.11}$$

the set of message vectors $m$ which satisfy that $J_{b+1}(m)$ is consisted of consecutive integers.

We now provide a lemma which states that within $M_{\text{gather}}(q)$, there exists a minimizer of the optimization problem (F.10).

**Lemma F.1.** For arbitrary $q \in \{0, 1, \ldots, L\}$, we have

$$v_q^* = \min_{m \in M_{\text{gather}}(q)} v(m).$$

**Proof.** From Fig. 9 and the definition of $v_q^*$, all we need to prove is the following statement: for all $m \in M(q) \cap (M_{\text{gather}}(q))^c$, we can assign another message vector $m^* \in M_{\text{gather}}(q)$ such that $v(m^*) \leq v(m)$ holds. Consider arbitrary $m \in M(q) \cap (M_{\text{gather}}(q))^c$, denoted as $m = [m_1, m_2, \ldots, m_n]$. Then, there exist integers $j \in \{1, \ldots, L\}$ and $\delta \in \{2, 3, \ldots, L - j\}$ such that $s + j, s + j + \delta \in J_{b+1}(m)$

Note that (F.9) implies (F.8), when the condition part is restricted to $|J_{b+1}(m)| = q$. 

---

**Figure 9:** Sets of message vectors used in proving Lemmas F.1 and F.2.
and \( s + j + 1, \ldots, s + j + \delta - 1 \notin J_{b+1}(m) \) hold. Select the smallest \( j \) which satisfies the condition. Consider \( m' = [m'_1, \ldots, m'_n] \) generated as the following rule:

1) The first \( s + j(b + 1) \) elements (which affect the first \( j \) rows of \( A \) in Figure 3) of \( m' \) is identical to that of \( m \).

2) The last \( n - (j + \delta - 1)(b + 1) - s \) elements of \( m \) are shifted to the left by \( (\delta - 1)(b + 1) \), and inserted to \( m' \). In the shifting process, we have \( b \) locations where the original \( m_i \) and the shifted \( m_i + (\delta - 1)(b + 1) \) overlap. In such locations, \( m'_i \) is set to the maximum of two elements; if either one is 1, we set \( m'_i = 1 \), and otherwise we set \( m'_i = 0 \).

This can be mathematically expressed as below:

\[
m'_i = \begin{cases} 
  m_i, & 1 \leq i \leq s + j(b + 1) \\
  \max\{m_i, m_{i+(\delta-1)(b+1)}\}, & s + j(b + 1) + 1 \leq i \leq s + (j + 1)(b + 1) \\
  m_{i+(\delta-1)(b+1)}, & s + (j + 1)(b + 1) + 1 \leq i \leq n - (\delta - 1)(b + 1) \\
  0, & n - (\delta - 1)(b + 1) + 1 \leq i \leq n 
\end{cases} \tag{F.12}
\]

Note that we have

\[
\sum_{i=1}^{n} m_i = \frac{n - 1}{2} \tag{F.13}
\]

since \( m \in M_{(n-1)/2} \). Moreover, (F.12) implies

\[
\sum_{i=1}^{n} m'_i = \sum_{i=1}^{s+j(b+1)} m'_i + \sum_{i=s+j(b+1)+1}^{s+(j+1)(b+1)} m'_i + \sum_{i=s+(j+1)(b+1)+1}^{n-(\delta-1)(b+1)} m'_i + \sum_{i=s+(\delta-1)(b+1)+1}^{n} m_i
\]

\[
= \sum_{i=1}^{s+j(b+1)} m_i + \sum_{i=s+j(b+1)+1}^{s+(j+1)(b+1)} m'_i + \sum_{i=s+(j+\delta)(b+1)+1}^{n} m_i
\]

\[
\geq n - 1 - \left( \sum_{i=s+j(b+1)+1}^{s+(j+\delta)(b+1)} m_i - \sum_{i=s+j(b+1)+1}^{s+(j+1)(b+1)} m'_i \right) \overset{(a)}{\geq} \frac{n - 1}{2} \tag{F.14}
\]
where Eq. (a) is from
\[ \sum_{i=s+j(b+1)+1}^{s+(j+\delta)(b+1)} m_i \geq \sum_{i=s+j(b+1)+1}^{s+(j+1)(b+1)} (m_i + m_{i+(\delta-1)(b+1)}) \geq \sum_{i=s+j(b+1)+1}^{s+(j+1)(b+1)} \max\{m_i, m_{i+(\delta-1)(b+1)}\} \]
\[ \text{F.12} \]

and Eq. (b) is from \( \delta \geq 2 \). Note that
\[ v(m') = v(m) - \epsilon \quad \text{(F.15)} \]

holds for
\[ \epsilon := \frac{n-1}{2} - \sum_{i=1}^{n} m'_i, \quad \text{(F.16)} \]

which is a non-negative integer from (F.14). Now, we show the behavior of \( J_{b+1}(m) \) as follows. Recall that for \( j_0 \in \{s + 1, \ldots, s + L\} \),
\[ G(j_0, i_0) = \begin{cases} 1, & \text{if } s + (j_0 - s - 1)(b + 1) + 1 \leq i_0 \leq s + (j_0 - s - 1)(b + 1) + 2b + 1 \\ 0, & \text{otherwise} \end{cases} \quad \text{(F.17)} \]

holds from Algorithm 1 and Fig. 3. Define
\[ J_+ := \{j' \in \{s + j + \delta, \ldots, s + L\} : j' \in J_{b+1}(m)\}, \]
\[ J_- := \{j' \in \{s + 1, \ldots, s + j\} : j' \in J_{b+1}(m)\}. \]

From (F.12) and (F.17), we have
\[ \begin{cases} J_- \subseteq J_{b+1}(m'), \\ \text{if } j' \in J_+, \text{ then } j' - (\delta - 1) \in J_{b+1}(m'). \end{cases} \]

Thus, we have
\[ |J_{b+1}(m')| \geq |J_-| + |J_+| = |J_{b+1}(m)| \overset{(a)}{=} q \quad \text{(F.18)} \]

where Eq. (a) is from \( m \in M(q) \).
Now we construct \( \mathbf{m}'' \in M^{(q)} \) which satisfies \( v(\mathbf{m}'') \leq v(\mathbf{m}) \). Define \( S_0 := \{ i \in [s] : m_i' = 0 \} \) and \( \epsilon_0 := |S_0| \). The message vector \( \mathbf{m}'' = [m''_1, \ldots, m''_n] \) is defined as follows.

**Case I** (when \( \epsilon \leq \epsilon_0 \)): Set \( m''_i = m'_i \) for \( i \in \{ s + 1, s + 2, \ldots, n \} \). Randomly select \( \epsilon \) elements in \( S_0 \), denoted as \( \{ i_1, \ldots, i_\epsilon \} = S_0^{(\epsilon)} \subseteq S_0 \). Set \( m''_i = 1 \) for \( i \in S_0^{(\epsilon)} \), and set \( m''_i = m'_i \) for \( i \in S_0 \setminus S_0^{(\epsilon)} \). Note that this results in

\[
v(\mathbf{m}'') = v(\mathbf{m}'). \tag{F.19}
\]

**Case II** (when \( \epsilon > \epsilon_0 \)): Set \( m''_i = 1 \) for \( i \in [s] \). Define \( S_1 := \{ i \in \{ s + 1, \ldots, n \} : m_i' = 0 \} \). Randomly select \( \epsilon - \epsilon_0 \) elements in \( S_1 \), denoted as \( \{ i'_1, \ldots, i'_{\epsilon - \epsilon_0} \} = S_1^{(\epsilon)} \subseteq S_1 \). Set \( m''_i = 1 \) for \( i \in S_1^{(\epsilon)} \), and set \( m''_i = m'_i \) for \( i \in \{ s + 1, \ldots, n \} \setminus S_1^{(\epsilon)} \). Note that this results in

\[
v(\mathbf{m}'') = v(\mathbf{m}') + (\epsilon - \epsilon_0). \tag{F.20}
\]

Note that in both cases, the weight of \( \mathbf{m}'' \) is

\[
||\mathbf{m}''||_0 = \sum_{i=1}^{n} m''_i = \sum_{i=1}^{n} m'_i + \epsilon \frac{n - 1}{2}. \tag{F.21}
\]

Moreover,

\[
|J_{b+1}(\mathbf{m}'')|^{(a)} \geq |J_{b+1}(\mathbf{m}')| \geq q \tag{F.22}
\]

holds where Eq.\((a)\) is from the fact that all elements of \( \mathbf{m}'' - \mathbf{m} \) are non-negative. Finally,

\[
v(\mathbf{m}'') = v(\mathbf{m}) - \min\{\epsilon, \epsilon_0\} \leq v(\mathbf{m}) \tag{F.23}
\]

holds from \((F.15), (F.19), \) and \((F.20)\). Combining \((F.21), (F.22)\) and \((F.23)\), one can confirm that \( \mathbf{m}'' \in M^{(q)} \) and \( v(\mathbf{m}'') \leq v(\mathbf{m}) \) hold; this gathering process\(^8\) maintains the weight of a message vector and does not increase the \( v \) value. Let \( \mathbf{m}^* \) be the message vector generated by applying this gathering process to \( \mathbf{m} \) sequentially until \( J_{b+1}(\mathbf{m}^*) \) is consisted of consecutive integers. Then, \( \mathbf{m}^* \) satisfies the followings:

1) \( J_{b+1}(\mathbf{m}^*) \) contains more than \( q \) elements. Moreover, since \( J_{b+1}(\mathbf{m}^*) \) is consisted of consecutive integers, we have \( \mathbf{m}^* \in M^{(q)}_{\text{gather}} \).

\(^8\)In Fig. \( 10 \) one can confirm that \( J_{b+1}(\mathbf{m}) \) is not consisted of consecutive integers (i.e., there's a gap), while \( J_{b+1}(\mathbf{m}'') \) has no gap. Thus, we call this process as gathering process.
We here prove that arbitrary message vector $m$ vector for someGiven a message vector $m$, since the above process of generating $G(s + 1:s + L, s + 1:n) = \begin{bmatrix} \text{Locations where } m_i = 1 \end{bmatrix} j = s + 1 \in J_{b+1}(m)$ \begin{bmatrix} \text{Locations where } m''_i = 1 \end{bmatrix} j = s + 2 \in J_{b+1}(m'')$

Figure 10: The gathering process illustrated in the proof of Lemma F.1, when $n = 17, b = 2$. Under this setting, we have $J_{b+1}(m) = J_3(m) = \{j \in \{s + 1, \ldots, s + L\} : m_T G(j,:) \geq 3\}$. Before the gathering process, we have $J_{b+1}(m) = \{s + 1, s + 3\}$, while $J_{b+1}(m'') = \{s + 1, s + 2\}$ holds after the process.

2) $v(m') \leq v(m'') \leq v(m)$ holds.

Since the above process of generating $m^* \in M_{\text{gather}}^{(q)}$ is valid for arbitrary message vector $m \in M^{(q)} \cap (M_{\text{gather}}^{(q)})^c$, this completes the proof. \hfill \Box

Now consider arbitrary message vectors satisfying $m \in M_{\text{gather}}^{(q)}$. Then, we have

$$J_{b+1}(m) = \{j, j + 1, \ldots, j + \delta - 1\}$$

(F.24)

for some $j \in \{s + 1, \ldots, s + L - \delta + 1\}$ and $\delta \geq q$. Here, we define

$$M_{\text{gather, overlap}}^{(q)} = \left\{ m \in M_{\text{gather}}^{(q)} : m_{s + (j_0 - s) + (b + 1)} = 0 \text{ for } j_0 \in \{j, \ldots, j + \delta - 1\} \right\}$$

(F.25)

We here prove that arbitrary message vector $m \in M_{\text{gather}}^{(q)}$ can be mapped into another message vector $m' \in M_{\text{gather, overlap}}^{(q)}$ without increasing the corresponding $v$ value, i.e., $v(m') \leq v(m)$. Given a message vector $m \in M_{\text{gather}}^{(q)}$, define $m' = [m'_1, m'_2, \ldots, m'_n]$ as in Algorithm 2. In line 9 of this algorithm, we can always find $l \in [s]$ that satisfies $m_l = 0$, due to the following reason.
Note that
\[ \sum_{i=s+1}^{n} m_i \overset{(a)}{=} m^T G(j,:) \overset{(F.24)}{\geq} b + 1 \] holds where \((a)\) is from the fact that \(G(j,i) = 0\) for \(i \in [s]\) as in \((F.17)\). Thus, we have
\[ \sum_{i=1}^{s} m_i \overset{(F.24)}{\leq} \sum_{i=1}^{n} m_i - (b + 1) = \frac{n - 1}{2} - (b + 1) = s - 1. \] Therefore, there exists \(l \in [s]\) such that \(m_l = 0\).

The vector \(m'\) generated from Algorithm 2 satisfies the following four properties:

1) \(m' \in M_{(n-1)/2}\),
2) \(J_{b+1}(m') = J_{b+1}(m) = \{j, j + 1, \ldots, j + \delta - 1\}\),
3) \(m' \in M_{\text{gather,overlap}}^{(q)}\),
4) \(v(m') \leq v(m)\).

The first property is from the fact that lines 7 and 10 of the algorithm maintains the weight of the message vector to be \(\|m\|_0 = (n - 1)/2\). The second property is from the fact that \((m')^T G(j_0,:) \overset{(F.17)}{=} \sum_{i=1}^{2b+1} m'_{s+(j_0-s-1)(b+1)+i} \overset{(a)}{=} \begin{cases} \sum_{i=1}^{2b+1} m_{s+(j_0-s-1)(b+1)+i}, & \text{if line 6 of Algorithm 2 is satisfied} \overset{(F.24)}{\geq} b + 1 \\ 2b, & \text{otherwise} \end{cases}\) for \(j_0 \in \{j, j + 1, \ldots, j + \delta - 1\}\), where \((a)\) is from the fact that \(\sum_{i=1}^{2b+1} m_{s+(j_0-s-1)(b+1)+i} = 2b + 1\) holds if line 6 of Algorithm 2 is not satisfied. The third property is from the first two properties and the definition of \(M_{\text{gather,overlap}}^{(q)}\) in \((F.25)\). The last property is from the fact that 1) each execution of line 7 in the algorithm maintains \(v(m') = v(m)\), and 2) each execution of line 10 in the algorithm results in \(v(m') = v(m) - 1\). Thus, combining with Lemma \(\text{F.1}\), we have the following lemma:

**Lemma F.2.** For arbitrary \(q \in \{0, 1, \ldots, L\}\), we have

\[ v^*_q = \min_{m \in M_{\text{gather,overlap}}^{(q)}} v(m). \]
Algorithm 2 Defining $m' \in M_{\text{gather,overlap}}^{(q)}$ from arbitrary $m \in M_{\text{gather}}^{(q)}$

1: Input: message vector $m = [m_1, m_2, \cdots, m_n]$ having $J_{b+1}(m) = \{j, j + 1, \cdots, j + \delta - 1\}$
2: Output: message vector $m' = [m'_1, m'_2, \cdots, m'_n]
3: Initialize: $m' = m$

4: for $j_0 = j$ to $j + \delta - 1$ do
5:   if $m_{s+(j_0-s)(b+1)+1} = 1$ then
6:     if $\exists i \in [2b+1]$ such that $m_{s+(j_0-s-1)(b+1)+i} = 0$ then
7:       Set $m'_{s+(j_0-s)(b+1)+i} = 1$, and set $m_{s+(j_0-s)(b+1)+1} = 0$.
8:     else
9:       Find $l \in [s]$ such that $m_l = 0$ (The existence of such $l$ is proven in explanation near (F.27).)
10:      Set $m'_l = 1$, and set $m'_{s+(j_0-s)(b+1)+1} = 0$.
11:   end if
12: end if
13: end for

\[ J_{b+1}(m) = \begin{array}{cccccc}
    j & j+1 & \vdots & \vdots & \vdots & \vdots \\
    j & j+1 & \vdots & \vdots & \vdots & \vdots \\
    j & j+1 & \vdots & \vdots & \vdots & \vdots \\
    j & j+1 & \vdots & \vdots & \vdots & \vdots \\
    j & j+1 & \vdots & \vdots & \vdots & \vdots \\
    j & j+1 & \vdots & \vdots & \vdots & \vdots \\
\end{array} \]

\[ a_l = \sum_{i=1}^{b} m_{s+(j-s-l)(b+1)+i}, \] (F.29)

According to Lemma [F.2], in order to find $v_*^q$, all that remains is to find the optimal $m \in M_{\text{gather,overlap}}^{(q)}$ which has the minimum $v(m)$. Consider arbitrary $m \in M_{\text{gather,overlap}}^{(q)}$ and denote

\[ J_{b+1}(m) = \{j, j + 1, \cdots, j + \delta - 1\}. \] (F.28)

Define the corresponding assignment vector $\{a_l\}_{l=0}^{\delta}$ as

\[ a_l = \sum_{i=1}^{b} m_{s+(j-s-l)(b+1)+i}, \] (F.29)

which represents the number of indices $i$ satisfying $m_i = 1$ within $l$th overlapping region, as
Then, we have
\[ a_{j_0} + a_{j_0+1} = \sum_{i=1}^{b} m_{s+(j_0-s-1)(b+1)+i} + m_{s+(j_0-s)(b+1)+i} \]
\[ \geq b + 1 \quad (\text{F.30}) \]
for \( j_0 \in \{j, j+1, \ldots, j+\delta-1\} \). Since \( a_t \) is the sum of \( b \) binary elements, we have
\[ 1 \leq a_t \leq b, \quad \forall t \in \{0, 1, \ldots, \delta\} \quad (\text{F.31}) \]
from (F.30). Now define a message vector \( m' \in M_{\text{gather, overlap}}^{(q)} \) satisfying the followings: the corresponding assignment vector is
\[ (a'_{0}, a'_{1}, \ldots, a'_{\delta}) = \begin{cases} (1, b, 1, b, \ldots, 1, b), & \text{if } \delta \text{ is odd} \\ (1, b, 1, b, \ldots, 1), & \text{otherwise}, \end{cases} \quad (\text{F.32}) \]
for \( a'_{t} = \sum_{i=1}^{b} m'_{s+(j-s-1+t)(b+1)+i} \), and the elements \( m'_{i} \) for \( i \in [s] \) is \( m'_{i} = 1_{\{i \leq i_{\max}\}} \) where \( i_{\max} = \frac{n-1}{2} - \sum_{l=0}^{\delta} a'_{l} \leq s \). Then, we have
\[ v(m) \geq \sum_{l=0}^{\delta} a_{l} \geq \sum_{l=0}^{\delta} a'_{l} v(m') \quad (\text{F.33}) \]
for arbitrary \( m \in M_{\text{gather, overlap}}^{(q)} \). Moreover, among \( \delta \geq q \), setting \( \delta = q \) minimizes \( v(m') \), having the optimum value of
\[ v^{*}_{q}(a)(m') \stackrel{\text{(a)}}{=} v(m') \geq \begin{cases} \sum_{i=1}^{q+1} (1 + b), & \text{if } q \text{ is odd} \\ 1 + \sum_{i=1}^{q} (1 + b), & \text{otherwise} \end{cases} \]
\[ \geq \begin{cases} 1 + b + 2 \frac{q+1}{2} - 1 = b + q, & \text{if } q \text{ is odd} \\ 1 + (1 + b) + 2 \frac{q}{2} - 1 = b + q, & \text{otherwise} \end{cases} \]
where (a) is from (F.33) and Lemma (F.2) and (b) is from \( b \geq 1 \). Combining this with the definition of \( v_{q}^{*} \) in (F.10) proves (F.9). This completes the proofs for (F.3) and (F.1). Thus, the data allocation matrix \( G \) in Algorithm 1 perfectly tolerates \( b \) Byzantines. From Fig. 3 the required
redundancy of this code is

\[ r^{(u)} = \frac{s + (2b + 1)L + n(n - s - L)}{n} = \left( n - (2b + 1) \right) \frac{2}{2n} - \frac{2b + 1}{2n} L + \left( \frac{n + (2b + 1)}{2} - L \right) \]

where Eq. (a) is from the definition of \( s \) in Algorithm 1.

\[ \text{APPENDIX G} \]

\[ \text{PROOF OF THEOREM 4} \]

**Proof.** Note that \( r^* \leq r^{(u)} \) is obtained directly from Theorem 3 and the definition of \( r^* \). Thus, all that remains is to show \( r^* \geq r^{(l)} \). For a given allocation matrix \( G \in \{0, 1\}^{n \times n} \), define

\[ a_l := \sum_{l=1}^{n} 1 \{ \| G(j,:) \|_0 = l \} \]

Since \( n \) and \( \| G(j,:) \|_0 \) are assumed to be an odd number, we have

\[ a_2 = a_4 = \cdots = a_{n-1} = 0. \]

Thus, we have

\[ n = \sum_{l=1}^{n} a_l = a_1 + a_3 + \cdots + a_{n-2} + a_n \]  \hspace{1cm} (G.1)

where Eq. (a) is from \( 1 \leq \| G(j,:) \|_0 \leq n \). The redundancy factor of matrix \( G \) can be written as

\[ r = \left( \text{\# of 1's in } G \right) \frac{n}{n} = a_1 + 3a_3 + \cdots + na_n \]

\[ = \frac{n(a_1 + a_3 + \cdots + a_n) - \{(n-1)a_1 + (n-3)a_3 + \cdots + 2a_{n-2} + 0 \cdot a_n\}}{n} \]

\[ n - \frac{(n-1)a_1 + (n-3)a_3 + \cdots + 2a_{n-2}}{n} \]  \hspace{1cm} (G.2)

For a given assignment vector defined as \( a = (a_1, a_3, \cdots, a_n) \), we denote the redundancy as \( r = r(a) \). Recall that according to Theorem 2 in the main manuscript, a system with matrix \( G \) is perfect \( b \)-Byzantine tolerable if it satisfies

\[ \sum_{v=1}^{n-1} |J_v(m)| \leq \frac{n-1}{2} - b \]  \hspace{1cm} (G.3)

for all \( m \in M_{(n-1)/2} \). Thus, whether a system with data allocation matrix \( G \) satisfies the perfect \( b \)-Byzantine tolerance of a system is determined by the corresponding assignment vector \( a = (a_1, a_3, \cdots, a_n) \). Now we prove that constraints in Theorem 4 are the necessary conditions
on $\{a_{2t-1}\}_{t=1}^{(n-1)/2}$ for perfect $b$–Byzantine tolerance. Note that the last two constraints in Theorem 4 are trivial necessary conditions. Thus, proving for the first two inequalities is enough. First, we prove $a_1 \leq \frac{n-1}{2} - b$ is required for perfect $b$–Byzantine tolerance. Suppose $a_1 > \frac{n-1}{2} - b$.

Denote the set of $j \in [n]$ that satisfies $\|G(j,:))_0 = 1$ as $\{j_1, j_2, \cdots, j_{a_1}\}$. Moreover, for $p \in [a_1]$, define $i_p$ as the unique integer satisfying $G(j_p, i_p) = 1$. Consider an arbitrary $m \in M_{(n-1)/2}$ satisfying $m_i = 1$ for $i \in \{i_1, i_2, \cdots, i_{a_1}\}$. Then, $m^TG(j,:)) \geq m_{i_p}G(j_p, i_p)$ holds for $p \in [a_1]$, which results in $j_1, j_2, \cdots, j_{a_1} \in J_1(m)$. Thus, we have

$$\sum_{v=1}^{(n-1)/2} |J_v(m)| \geq |J_1(m)| = a_1 > \frac{n-1}{2} - b.$$  

Therefore, if $a_1 > (n - 1)/2 - b$, then the system is not perfect $b$–Byzantine tolerable. This implies that $a_1 \leq (n - 1)/2 - b$ is a necessary condition for perfect $b$–Byzantine tolerance.

Now we prove that the second constraint in Theorem 4 is a necessary condition for perfect $b$–Byzantine tolerance. Suppose a system with allocation matrix $G$ is perfect $b$–Byzantine tolerable. Then,

$$\sum_{j=1}^{n} \sum_{m \in M_{n-1}} 1\{m^TG(j,:) > \frac{\|G(j,:))_0}{2}\} = \sum_{m \in M_{n-1}} \sum_{j=1}^{n} 1\{m^TG(j,:) > \frac{\|G(j,:))_0}{2}\}$$

$$\overset{(a)}{=} \sum_{m \in M_{n-1}} \sum_{v=1}^{n+1} |J_v(m)| \overset{(b)}{=} \sum_{m \in M_{n-1}} \sum_{v=1}^{n+1} |J_v(m)| \leq |M_{n-1}| \left(\frac{n-1}{2} - b\right) = \left(\frac{n-1}{2}\right) \left(\frac{n-1}{2} - b\right). \quad (G.4)$$

Here, $(a)$ is from the fact that for each $j \in [n]$, there exists a unique $v \in \{1, 2, \cdots, \frac{n+1}{2}\}$ with $\|G(j,:))_0 = 2v - 1$. Moreover, $(b)$ is from the fact that $J_{\frac{n+1}{2}}(m)$ is an empty set, which is obtained by the definition of $J_v(m)$ in [10] and the fact that $\|m\|_0 = (n - 1)/2$. Now define

$$x_j := \sum_{m \in M_{n-1}} 1\{m^TG(j,:) > \frac{\|G(j,:))_0}{2}\}.$$  

Consider an arbitrary $j \in [n]$. We have $\|G(j,:))_0 = 2v - 1$ for some $v \in \{1, 2, \cdots, \frac{n+1}{2}\}$. Then,
the number of message vectors \( m \in M_{n-1} \) satisfying \( m^T G(j,:) > \frac{\|G(j,:)\|_0}{2} \) can be written as

\[
x_j = \begin{cases} 
  \sum_{i=v}^{\min\{2v-1, \frac{n+1}{2}\}} \left( \begin{array}{c} 2v-1 \\ i \\ \end{array} \right) \left( \begin{array}{c} n-2v+1 \\ \frac{n+1}{2} - i \\ \end{array} \right), & \text{if } v \leq \frac{n-1}{2} \\
  0, & \text{if } v = \frac{n+1}{2}
\end{cases}
\]

for \( v = \frac{\|G(j,:)\|_0+1}{2} \). Since the number of \( j \in [n] \) satisfying \( \|G(j,:)\|_0 = 2v - 1 \) is \( a_{2v-1} \), the left-hand-side of (G.4) can be written as

\[
\text{LHS of (G.4)} = \sum_{j=1}^{n} x_j = \sum_{v=1}^{n-1} a_{2v-1} \sum_{i=v}^{\min\{2v-1, \frac{n+1}{2}\}} \left( \begin{array}{c} 2v-1 \\ i \\ \end{array} \right) \left( \begin{array}{c} n-2v+1 \\ \frac{n+1}{2} - i \\ \end{array} \right).
\]

Thus, combining the above equation with (G.4), we obtain the second necessary condition in Theorem 4. This proves that the four constraints in Theorem 4 is a necessary condition for perfect \( b \)-Byzantine tolerance.

Now, let \( \mathcal{A} \) be the set of assignment vectors \( a = (a_1, a_3, \cdots, a_n) \) satisfying the perfect \( b \)-Byzantine tolerance. Moreover, denote \( \mathcal{A}_{\text{relax}} \) be the set of of assignment vectors \( a \) satisfying the four conditions in Theorem 4. Since the four conditions in Theorem 4 are the necessary conditions on the perfect \( b \)-Byzantine tolerance, we have \( \mathcal{A} \subseteq \mathcal{A}_{\text{relax}} \). Thus, we have

\[
\min_{a \in \mathcal{A}} r(a) \geq \min_{a \in \mathcal{A}_{\text{relax}}} r(a) \overset{(G.2)}{=} n - \frac{z}{n} = r^{(l)},
\]

which completes the proof.

\[ \square \]

\section*{References}

[1] T. Ben-Nun and T. Hoefler, “Demystifying parallel and distributed deep learning: An in-depth concurrency analysis,” ACM Computing Surveys (CSUR), 2019.

[2] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le et al., “Large scale distributed deep networks,” in Advances in neural information processing systems (NIPS), 2012, pp. 1223–1231.

[3] 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.

[4] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard et al., “Tensorflow: A system for large-scale machine learning,” in 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016, pp. 265–283.

[5] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, “Automatic differentiation in pytorch,” 2017.
[6] F. Seide and A. Agarwal, “Cntk: Microsoft’s open-source deep-learning toolkit,” in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2016, pp. 2135–2135.

[7] B. Recht, C. Re, S. Wright, and F. Niu, “Hogwild: A lock-free approach to parallelizing stochastic gradient descent,” in Advances in neural information processing systems, 2011, pp. 693–701.

[8] H. B. McMahan, E. Moore, D. Ramage, S. Hampson et al., “Communication-efficient learning of deep networks from decentralized data,” arXiv preprint arXiv:1602.05629, 2016.

[9] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, “Federated learning: Strategies for improving communication efficiency,” arXiv preprint arXiv:1610.05492, 2016.

[10] 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.

[11] D. Alistarh, Z. Allen-Zhu, and J. Li, “Byzantine stochastic gradient descent,” in Advances in Neural Information Processing Systems, 2018, pp. 4613–4623.

[12] E. M. El Mhamdi, R. Guerraoui, and S. Rouault, “The hidden vulnerability of distributed learning in Byzantium,” in Proceedings of the 35th International Conference on Machine Learning, 2018, pp. 3521–3530.

[13] E.-M. El-Mhamdi and R. Guerraoui, “Fast and secure distributed learning in high dimension,” 2019.

[14] 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.

[15] 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.

[16] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, “Byzantine-robust distributed learning: Towards optimal statistical rates,” in Proceedings of the 35th International Conference on Machine Learning, 2018, pp. 5650–5659.

[17] L. Chen, H. Wang, Z. Charles, and D. Papailiopoulos, “Draco: byzantine-resilient distributed training via redundant gradients,” in Proceedings of the 35th International Conference on Machine Learning, 2018, pp. 902–911.

[18] H. Wang, S. Sievert, S. Liu, Z. Charles, D. Papailiopoulos, and S. Wright, “Atomo: Communication-efficient learning via atomic sparsification,” in Advances in Neural Information Processing Systems, 2018, pp. 9850–9861.

[19] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, “Deep gradient compression: Reducing the communication bandwidth for distributed training,” in International Conference on Learning Representations (ICLR-18), 2018.

[20] J. Wu, W. Huang, J. Huang, and T. Zhang, “Error compensated quantized SGD and its applications to large-scale distributed optimization,” in Proceedings of the 35th International Conference on Machine Learning, 2018, pp. 5325–5333.

[21] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, “signSGD: Compressed optimisation for non-convex problems,” in Proceedings of the 35th International Conference on Machine Learning, 2018, pp. 560–569.

[22] J. Bernstein, J. Zhao, K. Azizzadenesheli, and A. Anandkumar, “signSGD with majority vote is communication efficient and fault tolerant,” in International Conference on Learning Representations (ICLR-18), 2019.

[23] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, “Qsgd: Communication-efficient sgd via gradient quantization and encoding,” in Advances in Neural Information Processing Systems, 2017, pp. 1709–1720.

[24] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, “Terngrad: Ternary gradients to reduce communication in distributed deep learning,” in Advances in neural information processing systems, 2017, pp. 1509–1519.

[25] S. P. Kairam, Q. Rebjock, S. Stich, and M. Jaggi, “Error feedback fixes SignSGD and other gradient compression schemes,” in Proceedings of the 36th International Conference on Machine Learning, 2019, pp. 3252–3261.
[26] S. Rajput, H. Wang, Z. Charles, and D. Papailiopoulos, “Detox: A redundancy-based framework for faster and more robust gradient aggregation,” *arXiv preprint arXiv:1907.12205*, 2019.

[27] 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, 2018.

[28] K. Lee, C. Suh, and K. Ramchandran, “High-dimensional coded matrix multiplication,” in *2017 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2017, pp. 2418–2422.

[29] H. Park, K. Lee, J.-y. Sohn, C. Suh, and J. Moon, “Hierarchical coding for distributed computing,” in *2018 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2018.

[30] 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.

[31] S. Dutta, V. Cadambe, and P. Grover, “Short-dot: Computing large linear transforms distributedly using coded short dot products,” in *Advances In Neural Information Processing Systems*, 2016, pp. 2100–2108.

[32] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, “Gradient coding: Avoiding stragglers in distributed learning,” in *Proceedings of the 34th International Conference on Machine Learning*, 2017, pp. 3368–3376.

[33] W. Halbawi, N. Azizan, F. Salehi, and B. Hassibi, “Improving distributed gradient descent using reed-solomon codes,” in *2018 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2018, pp. 2027–2031.

[34] N. Raviv, R. Tandon, A. Dimakis, and I. Tamo, “Gradient coding from cyclic MDS codes and expander graphs,” in *Proceedings of the 35th International Conference on Machine Learning*, 2018, pp. 4305–4313.

[35] Z. Charles, D. Papailiopoulos, and J. Ellenberg, “Approximate gradient coding via sparse random graphs,” *arXiv preprint arXiv:1711.06771*, 2017.

[36] V. Conitzer and T. Sandholm, “Common voting rules as maximum likelihood estimators,” in *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, ser. UAI’05. Arlington, Virginia, United States: AUAI Press, 2005, pp. 145–152. [Online]. Available: [http://dl.acm.org/citation.cfm?id=3020336.3020354](http://dl.acm.org/citation.cfm?id=3020336.3020354)

[37] A. D. Procaccia, N. Shah, and Y. Zick, “Voting rules as error-correcting codes,” *Artificial Intelligence*, 2016.

[38] D. J. MacKay, “Fountain codes,” *IEEE Proceedings-Communications*, vol. 152, no. 6, pp. 1062–1068, 2005.

[39] K. G. Murty, *Linear programming*. Springer, 1983.

[40] L. D. Dalcin, R. R. Paz, P. A. Kler, and A. Cosimo, “Parallel distributed computing using python,” *Advances in Water Resources*, vol. 34, no. 9, pp. 1124 – 1139, 2011.

[41] M. Ye and E. Abbe, “Communication-computation efficient gradient coding,” in *Proceedings of the 35th International Conference on Machine Learning*, 2018, pp. 5610–5619.