Exact Fault-Tolerant Consensus with Voting Validity

Zhangchen Xu
Department of Electrical & Computer Engineering
University of Washington
Seattle, United States
zxu9@uw.edu

Yuetai Li
James Watt School of Engineering
University of Glasgow
Glasgow, United Kingdom
2510959L@student.gla.ac.uk

Chenglin Feng
James Watt School of Engineering
University of Glasgow
Glasgow, United Kingdom
2357707F@student.gla.ac.uk

Lei Zhang
James Watt School of Engineering
University of Glasgow
Glasgow, United Kingdom
Lei.Zhang@glasgow.ac.uk

***Abstract*—This paper investigates the multi-valued fault-tolerant distributed consensus problem that pursues exact output. To this end, the voting validity, which requires the consensus output of non-faulty nodes to be the exact plurality of the input of non-faulty nodes, is investigated. Considering a specific distribution of non-faulty votes, we first give the impossibility results and a tight lower bound of system tolerance achieving agreement, termination and voting validity. A practical consensus algorithm that satisfies voting validity in the Byzantine fault model is proposed subsequently. To ensure the exactness of outputs in any non-faulty vote distribution, we further propose safety-critical tolerance and a corresponding protocol that prioritizes voting validity over termination property. To refine the proposed protocols, we propose an incremental threshold algorithm that accelerates protocol operation speed. We also optimize consensus algorithms with the local broadcast model to enhance the protocol's fault tolerance ability.**

***Index Terms*—Fault Tolerance, Distributed Algorithms, Impossibility Results, Voting, Exact Consensus**

I. Introduction

A. Background: Consensus and Strong Validity

The distributed consensus studies how to guarantee the nodes agree on an identical binary value, despite the existence of faulty (i.e., crash or Byzantine) nodes and unreliable communication links. The implementation of consensus in distributed systems relies on *consensus algorithms* (aka consensus protocols), which must satisfy *termination*, *agreement* and *validity* properties [1]. Termination ensures the algorithm outputs a value in a finite time; agreement guarantees the states of nodes are identical; validity aims to ensure the output value is reasonable. Though defined differently in some literature, the most common definition of validity is: if all non-faulty nodes begin with the same input value, they output that value [2]. The above validity definition is a suitable constraint if the input domain is binary. However, if the input domain contains more than two values, the consensus output can be the input from a faulty node, which is hazardous. Therefore, the *Strong Consensus* problem was further proposed by Neiger et al. in [3] to solve this problem. As a *multi-valued* consensus [4], the strong consensus extends the node input from binary (e.g., $\{0,1\}$) to arbitrary (e.g., $\{0,1,2,3\}$). A strong consensus retains the termination and agreement properties and replaces the original Validity with *Strong validity* as shown below:

- **Termination** Every non-faulty node chooses a single output value and halts.
- **Agreement** The output value of non-faulty nodes are identical.
- **Strong Validity** The output value of each non-faulty node must be the input value of some non-faulty nodes.

Though the strong validity strengthens the original validity property, it might lack applied meaning, as the input of an arbitrary non-faulty node is regarded as a legitimate output. Therefore, further literature linked the strong consensus output with various physical meanings by proposing new validity definitions. For example, in [5], Stolz et al. introduced *median validity* by requiring the output to be close to the median of non-faulty inputs. They stated that if faulty nodes exist, no algorithm can guarantee to find the correct median of non-faulty nodes. In [6], Melnyk et al. proposed the concept of *interval validity* that accepts consensus values that are close to the k^{th} smallest value of non-faulty nodes. The proposed protocol is proved to be the best approximation to the desired value for synchronous systems.

B. Motivation

The motivation for our paper derives from the need for a reliable voting (election) mechanism based on distributed consensus. Note that although the idea of voting already plays a vital role in distributed computing [7], there is a conceptual difference between the term "voting" in distributed consensus and social choice. In the former field, voting is a mechanism that produces agreements among different nodes. Here, what matters more is to reach an agreement rather than what the agreement has reached, i.e., all nodes are assumed to have no specific preferences for one option [8]. On the contrary, voting in the social choice problem is a way for preference aggregation. Therefore, a natural idea is to crossover these concepts, i.e., try to ensure not only **agreement** but also realize **preference aggregation** in the consensus process.

Such crossover between social choice and distributed consensus can bring considerable benefits to applications in distributed systems and computational science, and we name a few here. For example, in a multi-agent distributed coordination problem [9] such as joint decision-making in connected autonomous driving, multiple vehicles in a cluster share information and decide on the following actions together. Distributed consensus algorithms are considered a viable solution to deal with this problem [10], [11]. However, applying the existing log-based consensus algorithms (e.g., PBFT [12] or Raft [13]) solely consent on a log entry proposed by an individual, which prevents non-faulty nodes from expressing their preferences. Therefore, the hazards of dictatorship may occur without a democratic voting mechanism. Another example of benefits is blockchain technology for edge computing, where the process of fork resolution [14] and leader election [15] can be refined by the distributed voting mechanism by providing a reliable and well-founded ruling process in the absence of mutual trust.

Inspired by strong validity and its variants, our design simultaneously achieves agreement and preference aggregation in distributed consensus context. Given that a consensus algorithm already satisfies the agreement property, we represent the preference aggregation in a validity definition, since various definitions of validity can give different physical meanings to the output. The new validity is defined as the output must be the exact plurality of the input values of non-faulty nodes. This follows the plurality voting scheme [16], where a winning candidate has more votes than other candidates, so the purpose of preference aggregation is revealed in the output. We call such validity as **Voting Validity**, and the formal definition will be given in Section III. Note that the mathematical meaning of voting validity is also similar to the mode of input values from non-faulty nodes, which is representative and is not affected by the extremely large or small values of a distribution series.

Note that voting validity requires the exactness of the output. However, realizing the exact output in a fault-tolerant distributed system for multi-valued consensus is challenging. The reason is that non-faulty nodes might be interfered with by faulty nodes' malicious behaviors, preventing the system from outputting the desired value. Consider a distributed system with 10 nodes, and 3 of them are Byzantine nodes. Assume the inputs of these seven non-faulty nodes are $\{0, 0, 0, 1, 1, 2, 3\}$. An output that satisfies voting validity must be 0, as it is the plurality in the input value set. However, if 3 Byzantine nodes collude and all vote for 1, there are 5 1s and only 3 0s from the perspective of a non-faulty node. Intuitively, the consensus output ought to be 1 instead of 0. The voting validity would then not be satisfied, and the exactness of the pursuit is lost.

Unfortunately, the exactness in previous works is absent, reflected in the weaker definition of termination, agreement, and the proposed validity. Some solutions weaken the agreement property by either allowing each node to output more than one value (i.e., the k-set consensus in [17]) or allowing each node to output a single value, different from each other but within a distance of ϵ (i.e., the approximate agreement [18]). On the other hand, some solutions retain the agreement but relax the validity by allowing the output with a legitimate interval. The example of median and interval validity shows that the identical output of each non-faulty node can only be reasonably close to the exact value, but cannot guarantee to reach it due to the impossibility results. Such good-enough output may make sense in some applications, for example, avoiding the interference of extreme values caused by sensor failure from the normal measured values in the sensor network [5]. However, since the exact value is unreachable, such algorithms might not be suitable for applications that require deterministic and accurate outputs in safety-critical scenarios (aircraft and vehicular control [19] [20], etc.). Unlike all of the above, we ensure the exactness of the output while maintaining both agreement and voting validity if certain conditions on the input values are met.

C. Our Contribution

Given that the exactness is our primary concern, we mainly focus on finding the condition (i.e., the lower bound on the number of nodes in relation to t and non-faulty input distribution) to ensure the voting validity in the presence of faulty nodes, and propose practical protocols that can be easily deployed on applications. The contributions of our paper are as follows.

1) We define voting validity that ensures the exact plurality of the output in the multi-valued consensus problem and gives the fault tolerance analysis of consensus with voting validity on both Byzantine fault and crash fault;
2) We design a practical synchronous Byzantine-fault-tolerant (BFT) protocol for the voting validity;
3) We consider a safety-critical case under the Byzantine fault model, and propose a protocol that ensures the consensus output (if it has) satisfies the voting validity with exactness, given arbitrary input values;
4) We define vote dispersion tolerance and analyze the system fault tolerance from a probability distribution perspective;
5) We propose practical voting protocols with an incremental threshold for optimistic responsiveness, and implement a local broadcast communication model to optimize the tolerance of existing protocols.

D. Roadmap

In the following paper, we give a literature review about distributed consensus and voting problems in Section II. Then in Section III, we present several model assumptions and related definitions. Section IV shows the impossibility results of voting validity under Byzantine and Crash faults. The corresponding BFT protocol is also proposed with proof of correctness. Section V considers the safety-critical scenario, where the exactness of the output is guaranteed. We propose safety-critical tolerance and give corresponding safety-guaranteed protocol in this section for such exactness. Section VI combines the lower bounds of voting validity under different failure cases, and the vote dispersion threshold is proposed with the probabilistic analysis. Section VII is for the optimization of consensus

protocols. Section VIII gives the conclusion of this paper and discusses potential future work.

II. Related Works

The fault-tolerant distributed consensus problem was first proposed in [21] and [22] by Lamport, Shostak and Pease. They showed that to tolerate Byzantine fault without setup assumptions (e.g., public key infrastructure) and digital signatures, the total number of nodes N must be greater than three times the number of the faulty nodes t, i.e., $N > 3t$. In contrast, a crash fault requires only $N > 2t$. Since then, the distributed consensus has been studied extensively under different assumptions, e.g., synchronous and asynchronous networks, shared memory and message-passing models [2].

Compared to classic binary consensus problems that only consider binary inputs, the concept of multi-valued consensus was proposed in [4], which showed how to build multi-valued consensus from binary consensus. In [3], a strong consensus with stricter output requirements (i.e., only inputs from non-faulty nodes are legitimate) was further proposed. [3] indicated that the strong consensus is not solvable if $N \leq mt$, where m is the size of the input domain. There were variants of strong consensus that consider the physical meanings and application values of the consensus output, such as in [23], [5], [6], [24] and [25]. Nevertheless, these agreements cannot guarantee to achieve the exact desired output value (e.g., the median value of the input) but can only be close to it. As we seek the exactness of distributed consensus (i.e., the voting validity), our analysis and results differ from all the above literature.

Our voting validity aims to find the plurality of non-faulty inputs as a means of preference aggregation. Therefore, it is related to democratic elections, and there are few prior works on consensus theory and social choice. In economic and game theory, there are two main forms of election: social choice function and social welfare function [26]. The difference is that the former outputs one candidate as the winner, while the latter outputs the ranking of the candidates. In [27], Chauhan et al. first considered election in faulty distributed systems and drew preliminary conclusions about social choice and social welfare problems respectively, considering Byzantine faults. Then considering the social welfare problem, Melnyk et al. proposed in [28] a deterministic algorithm to solve rankings under Pareto Validity. For the social choice problem, in [23], Fitzi et al. proposed the concept of *δ-differential consensus* that requires the number of occurrences (i.e., plurality) of any other input value cannot exceed the plurality of the output by δ. The result indicated that the δ-differential consensus is impossible if $n \leq 3t$ or $\delta < t$. Then in [29], the Byzantine social choice problem is further investigated. Two relaxed voting property (i.e., l-majority(plurality) and c-verified) is proposed to generate a good-enough output close to the best result, namely the majority of the input values. However, the output of the proposed algorithm is still approximate rather than exact. A recent work [8] discussed the importance of social choice in blockchain research and illustrated the potential for cross-fertilization between them.

III. Notations, Models and Definitions

This section presents symbolic notations, network assumptions and communication models. Then we give several frequently used definitions in the following sections.

A. Notations

There are in total N nodes in a distributed system. At most t of all nodes are faulty, which refers to the tolerance of the system, i.e., the maximum number of faulty nodes that enables the system to work normally. We denote the actual number of faulty nodes during the execution of the consensus algorithm as f ($f \leq t$). Others are called non-faulty nodes or good nodes. A non-faulty node does not know the value of f but knows t as a pre-determined global parameter.

The following notations are related to the voting scenario in this paper. There is a subject s that requires all nodes to vote. We call the input of a node i *node preference* and denote it as v_i. We use the calligraphy letter $\mathcal{X}$ to represent an *option*, which acts as a set containing the same node preference inputted by different nodes (e.g., there might be $\mathcal{A}, \mathcal{B}, \mathcal{C}$, etc. in one voting). The *voting option domain* $\mathbb{V}$ refers to a set containing all options, i.e., $\mathcal{X} \in \mathbb{V}$. Note that $\mathbb{V}$ might be pre-determined by the subject s or generated by all nodes' inputs. The size of $\mathbb{V}$ is denoted by $|\mathbb{V}| = m$, indicating the number of options.

For a node i, we denote $i \to \mathcal{A}$ as a *vote* from node i who supports option $\mathcal{A}$, which will be broadcasted to other nodes. Due to the distributed nature, the received messages of different nodes may not be identical, so we use $\mathcal{X}_i$ to represent received votes of $\mathcal{X}$ in node i. A *ballot* is a collection of the votes in one execution of voting. We denote $|\mathcal{A}_i| = A_i$ as the number of votes received by node i that support $\mathcal{A}$. Note that if node i is non-faulty, it may receive votes from both non-faulty nodes and faulty nodes, so $A_i = A_{Gi} + A_{Fi}$, where A_{Gi} represents the number of votes from non-faulty nodes and A_{Fi} denotes the number of votes from faulty nodes.

For example, assume 7 nodes (one is a Byzantine node) propose three candidates in a leadership election: Alice, Bob and Carol. Three non-faulty voters support Alice, two support Bob, one supports Carol, and the faulty node support Bob. Assume Alice is $\mathcal{A}$, Bob is $\mathcal{B}$, and Carol is $\mathcal{C}$, we have $\mathbb{V} = \{\mathcal{A}, \mathcal{B}, \mathcal{C}\}$ and $m = 3$. For a non-faulty node 1, it receives two non-faulty votes and one faulty vote to Bob. In this case, $B_1 = 3$, $B_{G1} = 2$, $B_{F1} = 1$ (though node 1 does not know what B_{G1} and B_{F1} are).

B. Models and Assumptions

1) Faulty Model: A node is faulty if it stops running normally. This paper covers the analysis of both *Crash Fault* and *Byzantine Fault*. A node with a crash fault abruptly stops working without resuming. In contrast, a node with a Byzantine fault can act arbitrarily, e.g., sending contradicting messages to other nodes or simply remaining silent. Considering voting scenarios, Byzantine nodes may also collude, bribe or lobby. To consider the worst-case scenario, we assume there is a strong adversary that controls all Byzantine nodes to maximally prevent reaching consensus, including but not limited to

delaying messages, selective forwarding, etc. Since Byzantine nodes may send different messages to different nodes, we denote $k \xrightarrow{i} \mathcal{A}$ as a malicious node k sending $\mathcal{A}$ to node i.

Regarding the concern of potential manipulation in the context of voting, we assume the node preferences of non-faulty nodes are *independent* in the consensus process, e.g., they do not engage in collusion, bribery and lobbying. Note that the possible manipulating acts of the malicious adversary would not affect the consensus result as long as $f \leq t$.

2) Network Assumption: Different network assumptions may lead to different tolerance and protocol design. There are three commonly agreed levels of network synchrony [2]: *synchronous*, *partially synchronous* and *asynchronous*. The Fischer-Lynch-Paterson (FLP) result [30] indicated that in asynchronous networks, even if only one process fails, no algorithm can guarantee that non-faulty processes reach the consensus. Therefore, the discussions in this paper and proposed protocols are mainly based on *synchronous networks*. As all nodes under the synchronous network will receive the same vote from a good node with a delay with upper bound δ_t, we say (eventually) $A_{Gi} = A_G$ for all i.

3) Communication Model: If not specified, the communication model is a point-to-point model (such as in [22]). In this case, a Byzantine node k can send $k \xrightarrow{i} \mathcal{A}$ to node i but $k \xrightarrow{j} \mathcal{B}$ to node j to prevent reaching a consensus. To refine the proposed algorithm in Section VII-B, a *local broadcast model* [31] will be used. Under the local broadcast model, any message is received identically by all the neighbors within the network. This implies a faulty node k can only send $k \to \mathcal{A}$ to any other nodes (i.e., it cannot send different votes to different nodes).

C. Definitions

Here we give several definitions frequently used in the following tolerance analysis.

Definition III.1 (Voting Preference)**.** We define Voting Preference $\mathcal{A} \succ \mathcal{B}$ as more non-faulty nodes support option $\mathcal{A}$ than $\mathcal{B}$ in one voting.

Note that the definition of voting preference focuses on inputs of non-faulty nodes rather than all nodes. For the case of equal votes from non-faulty nodes (e.g., $A_G = B_G$), we assume all nodes have agreed on an established rule to choose the identical one from the two tie options (i.e., ties are broken arbitrarily). To simplify, we assume that non-faulty nodes choose $\mathcal{B}$ if $A_G = B_G$. Mathematically, $\mathcal{A} \succ \mathcal{B} \iff A_G > B_G$.

The following definition constrains the behavior of non-faulty nodes.

Definition III.2 (Integrity of Non-faulty Nodes)**.** A non-faulty node will not output $\mathcal{A}$ if it sees $B_i \geq A_i$.

We then give the definition of voting validity as follows.

Definition III.3 (Voting Validity)**.** If $\mathcal{A} \succ \mathcal{X}$ for any other $\mathcal{X} \in \mathbb{V}$, then the output of each non-faulty node is $\mathcal{A}$.

Since voting validity compares the number of support from non-faulty nodes, the output must be the input of at least one non-faulty node. This implies if a consensus algorithm satisfies voting validity, it must satisfy the strong validity mentioned in Section I.

Then consider the existence of more than two options (i.e., $m \geq 3$), among which the top 2 choices are $\mathcal{A}$ and $\mathcal{B}$ respectively, and $\mathcal{A} \succ \mathcal{B}$. To simplify the following analysis, we denote a set of other options as $\mathcal{C}$, where $\mathcal{C} = \mathbb{V} \backslash \{\mathcal{A}, \mathcal{B}\}$. Here $\backslash$ is the operator of except. We also define the size of $\mathcal{C}$ as C, where

$$C = \sum_{\mathcal{X} \in \mathcal{C}} |\mathcal{X}|. \tag{1}$$

For consistency, $\mathcal{A}, \mathcal{B}, \mathcal{C}$ in the following sections follow the above preferences and definitions, if not specified.

IV. Distributed Consensus with Voting Validity

In this section, we discuss the properties of distributed consensus with the proposed voting validity. Since the crash fault is a mild case of Byzantine fault, we will first discuss the Byzantine fault case and then turn to crash fault. After that, we give a practical BFT consensus protocol that achieves voting validity.

A. Impossibility Results of Voting Validity

1) Byzantine Fault:

Theorem 1. *[29] There is no Byzantine fault-tolerant consensus algorithm that can guarantee the voting validity when $|\mathbb{V}| = m \geq 2$ and $t \geq 1$.*

Proof. Since a strong consensus is impossible with arbitrary inputs [3], voting validity is naturally impossible as it satisfies strong validity. □

Theorem 1 shows the impossibility of voting validity with Byzantine fault when $m \geq 2$, if no prior knowledge is given. In contrast, we consider the existence of prior knowledge of the distribution of non-faulty inputs in the following Lemma 2, which provides an impossibility theorem from a different perspective.

Lemma 2. *No consensus algorithm can achieve voting validity given $A_G - B_G \leq t$ considering Byzantine fault.*

Proof. To prove it by contradiction, we assume there is an algorithm $\mathbb{A}$ that achieves voting validity in $A_G - B_G \leq t$. Since a consensus algorithm also satisfies the agreement property, all non-faulty nodes should consent on $\mathcal{A}$ as $A_G > B_G$. However, there is a possibility that the number of Byzantine nodes $f = t$, and all Byzantine nodes deliberately vote for $\mathcal{B}$ instead. As a result $B_i = B_G + B_{Fi}$ for all non-faulty nodes. Recall that $A_G - B_G \leq t$, so $B_G = B_i - B_{Fi} \geq A_G - t$. As in this case $B_{Fi} = f = t$ and $A_i = A_G$, all non-faulty nodes see $B_i \geq A_i$. Recall the Integrity property of non-faulty nodes in Definition III.2, non-faulty nodes will not output $\mathcal{A}$ as consensus result, so we derive a contradiction, and the algorithm $\mathbb{A}$ cannot exist. □

It can be shown that when there is prior knowledge of input values, the impossibility result is relaxed, i.e., a large enough gap between A_G and B_G may be possible for the voting validity. The following Theorem 3 considers multiple options in one voting.

Theorem 3. *No consensus algorithm can achieve voting validity given $N \leq 2t + 2B_G + C_G$ considering Byzantine fault.*

Proof. From a global perspective, the total number of nodes N equals the number of Byzantine Nodes (f) plus the number of non-faulty nodes that support different options. Note that $f \leq t$, we have

$$N = A_G + B_G + C_G + f \leq A_G + B_G + C_G + t. \quad (2)$$

Substitute A_G in $A_G - B_G \leq t$ in Lemma 2 using Inequality (2), we can derive $N - B_G - C_G - t \leq A_G \leq B_G + t$, so $N \leq 2t + 2B_G + C_G$. □

Here we compare Theorem 3 with results of strong validity in [3]. Recall [3], consensus with strong validity is solvable only if $N > mt$, where m is the size of the input domain. The above result is derived based on the indistinguishability of Byzantine inputs and non-faulty inputs, and it did not take a priori of the non-faulty inputs into consideration. If we consider input distribution (i.e., A_G and B_G) in the context of strong consensus instead, we have the following property:

Property 1. *If $A_G > t$, then the strong validity holds.*

Proof. We prove it by contradiction. Assume the strong validity cannot be reached given $A_G > t$. In such case, the output must be an input of faulty nodes only, and there are $f > A_G$ faulty nodes inputted that value. However, this implies $f > t$, which is impossible. □

The above property shows the prior knowledge of A_G can relax the impossibility result of strong validity: given $A_G > t$, the strong validity is achievable even if $N < mt$. On the other hand, our voting validity is achievable only if $A_G - B_G > t$, so naturally $A_G > B_G + t > t$. This implies if voting validity holds, strong validity also holds.

2) Crash Fault: In a distributed system with crash fault only, although there are no malicious behaviors of Byzantine nodes, crashes can happen at any time in the round of execution. For example, a node may crash after it broadcasts an input. So counter-intuitively, $X_i \neq X_G$ can happen with crash fault.

Lemma 4. *No consensus algorithm can achieve voting validity given $A_G - B_G \leq t$ considering crash fault.*

Proof. Given those non-faulty nodes are pre-determined and any crash node can suddenly crash, we can have $X_i \neq X_G$ since a node may receive votes from a crash node. In the worst case that all t faulty nodes vote for $\mathcal{X}$ crash in one execution of the voting, $|X_i - X_G| = t$. Therefore, if such an algorithm exists, there is a possibility that $A_i - B_i = A_G - (B_G + t) \leq 0 \implies B_i \geq A_i$ for all non-faulty nodes. Recall Definition III.2, non-faulty nodes will not output $\mathcal{A}$ as consensus result. So there is a contradiction, and we conclude such a CFT algorithm does not exist. □

Theorem 5. *No consensus algorithm can achieve voting validity given $N \leq 2t + 2B_G + C_G$ considering crash fault.*

Proof. The proof is the same as in Theorem 3. □

For voting validity, BFT and CFT give identical impossibility results. The hidden reason is that in the limit, Byzantine nodes and crash nodes "sabotage" consensus (whether intentional or not) in the same way under the constraint of voting validity, i.e., they vote for the second-highest option. The Byzantine behaviors that send different votes to different nodes do not help to narrow the gap in votes. However, such Byzantine behaviors do affect the termination and agreement property of a BFT protocol, and this will soon be revealed in the following protocol design (i.e., there is a $3t$ in Inequality (3)).

B. BFT Voting Protocol

In this section, we design practical consensus protocols for voting validity. We only demonstrate the BFT protocol here due to space limitations, as BFT is a more general case than CFT. The proposed Algorithm 1 achieves termination, agreement and voting validity in a synchronous system given:

$$N > max\{3t, 2t + 2B_G + C_G\}. \quad (3)$$

1) The Protocol Design: There are two fundamental types of BFT protocols: *Byzantine Agreement* (BA) and *Byzantine Broadcast* (BB). In BA protocols, each node has an input value; while in BB, there is only one input value in a distinguished node (i.e., sender) who is responsible for sending the input to all nodes. A reliable BB protocol requires all non-faulty nodes to output the same value if the distinguished sender is non-faulty. A leader election mechanism exists if the sender is Byzantine. We believe in the voting scenario, it is more reasonable to have a sender who first broadcasts a subject that needs to be voted on, and then all nodes vote on it.

Therefore, we design a four-phase protocol, and the first phase uses a BB protocol (e.g., can be [32]) to reliably broadcast the subject in Line 6, i.e., guarantee the same subject in every non-faulty node. Then we propose a BA protocol to achieve voting validity in Line 7-17 of Algorithm 1. Given the above, we assign the role of all nodes as follows: one is called the *Speaker* who is responsible for collecting and broadcasting the subject s (i.e., the issue that needs to be voted on), and the others are members. After receiving the subject s from the speaker in Phase 1, all nodes broadcast their preferences about s in Phase 2. Note here the node preference v_i in Line 8 can be either one of an option contained in the subject or an arbitrary number. Voting option domain $\mathbb{V}_i$ in node i then keeps track of all possible options and stores the corresponding valid votes. Also, note that $\mathcal{A}$ in the protocol may not mean the same option in the perspective of each node, so when broadcasting its local highest-voted option, the node broadcasts its actual value.

Algorithm 1: BFT Voting Protocol

```
   /* Utilities                                          */
1  Function Sort(𝕍_i):
2      𝒜_i ← Highest-voted option;
3      ℬ_i ← Second-highest-voted option;
4      𝒞_i ← 𝕍_i\{𝒜_i, ℬ_i};
5      return 𝒜_i, ℬ_i, 𝒞_i;
   /* Main Protocol                                      */
   /* For Speaker                                        */
   ▷ Phase 1: Prepare Phase
6  ByzantineBroadcast(Subject s);
   /* For All Nodes                                      */
   ▷ Phase 2: Vote Phase
   /* Broadcast Node Preferences                         */
7  if Output subject s in Byzantine Broadcast then
8      Node Preference: v_i;
9      Broadcast(s, i → v_i);
   ▷ Phase 3: Propose Phase
   /* Receive and Sort Ballots                           */
10 if receiving at least t + 1 votes of s then
11     Wait for 2δ_t to receive all votes;
12     𝕍_i ← All votes received;
13     𝒜_i, ℬ_i, 𝒞_i = Sort(𝕍_i);
   /* Propose Option; For BFT, δ_P = 0                   */
14 if A_i − B_i > δ_P then
15     Broadcast(propose 𝒜_i);
   ▷ Phase 4: Decide Phase
16 if receiving at least N − t propose 𝒜_i then
17     output 𝒜_i;
```

In Algorithm 1, the BYZANTINEBROADCAST method refers to a generic BB protocol for providing the same subject s to all non-faulty nodes, e.g., [32]. By comparison, the BROADCAST method indicates sending messages to all nodes, which follows the point-to-point communication model defined in Section III. There is also a criterion for locally deciding which option to propose after receiving all valid votes. For any protocol in this paper, we define δ_P as the protocol's **local judgment condition**, i.e., a non-faulty node proposes $\mathcal{A}_i$ when $A_i - B_i > \delta_P$. Note that this condition may vary by protocol, and in Algorithm 1, $\delta_P = 0$. We will discuss the choice of the local judgment condition later in Section V.

2) Correctness of the Protocol: The lower bound $N > 3t$ of BB/BA [22] in the protocol's tolerance lower bound (Inequality (3)) mainly serves for the protocol's termination and agreement properties, and ensures reliable broadcast. Given that all non-faulty nodes have the same subject s, we now prove the correctness of Algorithm 1 as illustrated in Theorem 9. A consensus algorithm with voting validity is *correct* if it satisfies termination, agreement and voting validity. The following Lemma 6 (termination), Lemma 7 (agreement) and Lemma 8 (voting validity) prove the correctness of the protocol (Theorem 9).

Before presenting the proof, we provide the following property for the proof of Lemma 6 and Lemma 8.

Property 2. *If $A_G - B_G > t$, then $A_i > B_i$ for all non-faulty nodes.*

Proof. Recall a non-faulty node receives number of votes $A_i = A_G + A_{Fi}$, $B_i = B_G + B_{Fi}$, we have

$$\begin{aligned} A_i - B_i &= A_G - B_G + A_{Fi} - B_{Fi} \\ &> t + A_{Fi} - B_{Fi}. \end{aligned} \tag{4}$$

BFT voting follows $0 \le A_{Fi} \le t$, $0 \le B_{Fi} \le t$ and $0 \le A_{Fi} + B_{Fi} \le t$, so $-t \le A_{Fi} - B_{Fi} \le t$ for $\forall i$. Then from (4), we have $A_i - B_i > t + (-t) = 0$. Thus $A_i > B_i$. □

Lemma 6 (Termination)**.** *BFT Algorithm 1 achieves termination given $N > max\{3t, 2t + 2B_G + C_G\}$.*

Proof. Termination means every non-faulty node can output a value in a finite time if a non-faulty node proposes a value (i.e., the speaker is non-faulty). Since there are at least $N - t$ non-faulty nodes, Algorithm 1 can receive $\ge N - t$ votes in Line 10 and broadcast propose $\mathcal{A}_i$ in Line 15. In Property 2, we have shown if $A_G - B_G > t$, $A_i > B_i$ for all non-faulty nodes. Note that when substitute A_G in $A_G - B_G > t$ using Inequality (2), we have

$$N > 2t + 2B_G + C_G \Rightarrow A_G - B_G > t. \tag{5}$$

Therefore, all non-faulty nodes will propose $\mathcal{A}$ according to the integrity property in Line 15, which is exactly $N - t$ propose messages. Given the above, the condition in Line 16 is satisfied, and thus termination holds. □

Lemma 7 (Agreement)**.** *BFT Algorithm 1 achieves agreement given $N > max\{3t, 2t + 2B_G + C_G\}$.*

Proof. Since termination property is proved in Lemma 6, the proof of agreement property is equivalent to prove: if there are output $\mathcal{A}_i$ and $\mathcal{A}_j$ for any pair of two non-faulty nodes i and j, then $\mathcal{A}_i = \mathcal{A}_j$. To prove it by contradiction, if $\mathcal{A}_i \neq \mathcal{A}_j$ holds, there are at least $N - t$ nodes broadcasting $\mathcal{A}_i$ and $\mathcal{A}_j$ respectively. Since $2(N - t) - N = N - 2t > t$, there is a non-faulty overlap that broadcast both propose $\mathcal{A}_i$ and propose $\mathcal{A}_j$, which leads to a contradiction. Thus agreement holds. □

Lemma 8 (Voting Validity)**.** *BFT Algorithm 1 achieves voting validity given $N > 2t + 2B_G + C_G$.*

Proof. Since $N > 2t + 2B_G + C_G \Rightarrow A_G - B_G > t$ and the result from Property 2, all non-faulty nodes see $A_i > B_i$, propose $\mathcal{A}$ in Line 15 of Algorithm 1 and finally output $\mathcal{A}$. As $\mathcal{A}$ is the highest-voted option of non-faulty nodes, the output satisfies the voting validity. □

Theorem 9. *BFT Algorithm 1 achieves termination, agreement and voting validity given $N > max\{3t, 2t + 2B_G + C_G\}$.*

V. Safety-Guaranteed Protocol

Algorithm 1 can output consensus results with voting validity in synchronous distributed systems. However, if $N < 2t + 2B_G + C_G$, the small gap between A_G and B_G may enable Byzantine nodes to vote for the second-voted option of non-faulty nodes deliberately and make $B_i > A_i$ for all non-faulty nodes. Thus by the integrity definition, the voting validity cannot be guaranteed, and the exactness of the pursuit is also lost.

This case might be unacceptable in some safety-critical scenarios, especially if B_G is small, and all Byzantine nodes vote $\mathcal{B}$ to subvert the voting preference of good nodes. Given the above, we seek to improve the protocol and guarantee the voting validity for arbitrary votes from the non-faulty nodes as long as there is an output. This ensures exactness. We name such protocols **Safety-Guaranteed protocols** and the corresponding tolerance as **Safety-critical Tolerance (SCT)**. We mention here that there is a similar setting and method in quittable consensus [33] for failure detection. However, it may output Q (for quit) and violates the voting validity, so it cannot be applied in this case. By contrast, the proposed safety-guaranteed protocol guarantees voting validity, but may not meet the termination as a trade-off.

Definition V.1 (Safety-Guaranteed protocol)**.** Denote the output of all non-faulty nodes as v_o. We say a consensus protocol $\mathbb{A}$ with voting validity is a **safety-guaranteed protocol** if

$$\mathcal{A} \succ \forall_{\mathcal{X} \neq \mathcal{A}} \mathcal{X} \Rightarrow v_o \in \{\mathcal{A}, \emptyset\},$$

where $\emptyset$ is the state when the protocol does not terminate (e.g., non-faulty nodes fail to decide on the value in finite time). The safety-guaranteed protocol guarantees the output must satisfy the voting validity if the output is not an empty set.

A. Properties of Safety-Guaranteed Protocols

Here we demonstrate the properties of a safety-guaranteed protocol. Recall the definition of local judgment condition δ_P that $A_i - B_i > \delta_P$. The termination property requires this condition to hold for all non-faulty nodes. The following Property 3 indicates the relationship between $A_G - B_G$ and the local judgment condition δ_P.

Property 3. *If $A_G - B_G > \delta_P + t$, then $A_i - B_i > \delta_P$ for all non-faulty nodes.*

Proof. $A_i - B_i = (A_G - B_G) + (A_{Fi} - B_{Fi}) > (A_G - B_G)_{min} + (A_{Fi} - B_{Fi})_{min} > \delta_P + t + (-t) = \delta_P$. □

We then give an impossibility result of a safety-guaranteed protocol regarding its local judgment condition.

Theorem 10. *There is no safety-guaranteed protocol with a local judgment condition $\delta_P < t$ considering Byzantine fault.*

Proof. We prove this theorem by contradiction with indistinguishability. Assume a safety-guaranteed protocol $\mathbb{A}$ that outputs $\mathcal{A}$ with voting validity, and its local judgment condition $\delta_P < t$. Assume the number of votes received by a non-faulty node i satisfies $A_i - B_i = t > \delta_P$. Note that the local view of $A_i - B_i = t$ may result from different vote distributions of non-faulty nodes and various strategies of the Byzantine adversary, and we give two cases here. In case 1, $A_G > B_G$ and $t - (A_G - B_G)$ Byzantine nodes vote for $\mathcal{A}$. In this case, the output satisfies the voting validity is $\mathcal{A}$. In case 2, $A_G = B_G$, but all Byzantine nodes vote for $\mathcal{A}$. Since ties are broken arbitrarily, the output satisfies the voting validity can be $\mathcal{B}$. Given the same local view, a non-faulty node cannot distinguish the above two cases, and the output of either $\mathcal{A}$ or $\mathcal{B}$ is untenable. Therefore, the protocol $\mathbb{A}$ does not exist. □

Property 3 and Theorem 10 indicate the necessity of

$$A_G - B_G > \delta_P + t \geq 2t \tag{6}$$

for safety-guaranteed protocols. The Property 4 below indicates the global condition to satisfy the local judgment condition.

Property 4. *If $N > 3t + 2B_G + C_G$, then $A_i - B_i > t$ for all non-faulty nodes.*

Proof. Substitute N using Inequality (2), we have $A_G + B_G + C_G + t > 3t + 2B_G + C_G$, thus we have $A_G - B_G > 2t$. Then for all non-faulty nodes i, $A_i - B_i = (A_G - B_G) + (A_{Fi} - B_{Fi}) > 2t + A_{Fi} - B_{Fi}$. So $A_i - B_i > 2t + (A_{Fi} - B_{Fi})_{min} = 2t + (-t) = t$. □

B. The Protocol

Considering safety-critical scenarios, we present a safety-guaranteed protocol in Algorithm 2 with voting validity in a synchronous system given:

$$N > 3t + 2B_G + C_G. \tag{7}$$

Algorithm 2: Safety-Guaranteed BFT Voting Protocol (Change Phase 4 in Algorithm 1)

```
▷ Phase 3: Propose Phase
/* Receive and Sort Ballots                    */
16 if receiving at least t + 1 votes of s then
17 |  Wait for 2δ_t to receive all votes;
18 |  𝕍_i ← All votes received;
19 |  𝒜_i, ℬ_i, 𝒞_i = Sort(𝕍_i);
/* Propose Option; For SCT, δ_P = t            */
20 if A_i − B_i > δ_P then
21 |  Broadcast(propose 𝒜_i);
▷ Phase 4: Decide Phase
22 if receiving at least t + 1 propose 𝒜_i then
23 |  output 𝒜_i;
```

We do not demonstrate the full algorithm here, as Phase 1-2 between Algorithm 1 and 2 are identical. The differences are the alteration in the local judgment condition to $\delta_P = t$, and the change of the required number of propose $\mathcal{A}_i$ in the decide phase as shown above. Note that the number of propose in Line 22 drops to $t+1$ because $A_i - B_i > t \rightarrow A_G > B_G$, i.e.,

one broadcast from a non-faulty node can guarantee the voting validity. Theorem 11 proves the correctness of this protocol.

Theorem 11. *The safety-guaranteed consensus protocol in Algorithm 2 achieves termination, agreement and voting validity given $N > 3t + 2B_G + C_G$ considering Byzantine fault.*

Proof. The proof of termination property is similar to Lemma 6. As Property 4 indicates, all $N - f$ non-faulty nodes have $A_i - B_i > t$ given $N > 3t + 2B_G + C_G$, so $N - t \leq N - f$ propose $\mathcal{A}_i$ must be satisfied, and non-faulty nodes are always output $\mathcal{A}_i$ in finite time. Thus termination holds. For agreement, even if all $f \leq t$ Byzantine nodes propose another option $\mathcal{A}_j$, the number of proposed $\mathcal{A}_j$ cannot exceed t, so the agreement also holds. For voting validity, since $N > 3t + 2B_G + C_G \Rightarrow A_G - B_G > 2t$ in Property 4, the output $\mathcal{A}$ always satisfies the voting validity. □

To show the proposed Algorithm 2 is a safety-guaranteed protocol, we show the following property.

Property 5. *If $N \leq 3t + 2B_G + C_G$, then $v_o \in \{\mathcal{A}, \emptyset\}$.*

Proof. We can prove it by contradiction and assume there is an output $\mathcal{X} \notin \{\mathcal{A}, \emptyset\}$ when $N \leq 3t + 2B_G + C_G$. According to the definition, $X_G \leq A_G$. For a node i, $X_i \leq X_G + t$, so $X_i - A_i \leq t$ in any non-faulty nodes. Therefore, no non-faulty node broadcasts *propose* $\mathcal{X}_i$ in Line 15 of Algorithm 2, and the consensus cannot terminate with the output $\mathcal{X}$. □

Note that though the voting validity can always hold in Algorithm 1, the protocol may fail to terminate to the majority/plurality value in such a safety-critical case. Recall the results from the Byzantine fault in Section IV-A1, there is a trade-off in searching for exactness in consensus problems. Given the arbitrary input distribution of non-faulty nodes, termination and voting validity can not hold simultaneously in one protocol. Nevertheless, the safety-guaranteed protocol has its strength in some scenarios. In practice, the distributed system can conduct multiple rounds of votes to circumvent the situation of failing to terminate. Nodes can adjust their voting preferences (e.g., reconsider $\mathcal{A}$ and not vote for options in $\mathcal{C}$) to enlarge $A_G - B_G$ and allow the consensus to terminate successfully.

VI. Result Discussion

In this section, we summarize the results of Section IV and Section V using one inequality, and further introduce the concept of vote dispersion tolerance. We then discuss the voting validity from a probability point of view, and demonstrate the impossibility results of voting validity from the perspective of system entropy.

A. Vote Dispersion Tolerance

Recall Theorem 3 and Theorem 5, a protocol that achieves voting validity with both BFT and CFT should satisfy $N > 2t + 2B_G + C_G$, which yields $N/2 > t + (2B_G + C_G)/2$. Meanwhile, we show in Property 4 that $N > 3t + 2B_g + C_g$ is necessary for safety-guaranteed protocols, which yields $N/3 > t + (2B_G + C_G)/3$. As the inequalities of these three tolerance cases (i.e., BFT, CFT and SCT) have similar structures, we combine them together and present the following theorem.

Theorem 12. *The voting validity can hold if*

$$\frac{N}{K} > t + t_{vd}, \tag{8}$$

where $t_{vd} = (2B_G + C_G)/K$. For CFT and BFT, $K = 2$, and for SCT $K = 3$.

As t in the right side is widely recognized as the *fault tolerance* of a system where at most $f \leq t$ nodes can be faulty to reach a consensus, the remaining $t_{vd} = (2B_G + C_G)/K$ is not explicitly defined. We call this as term *vote dispersion tolerance*. To distinguish the traditional tolerance from ours, we call the right half of the inequality *system tolerance*, i.e., cannot exceed N/K considering both fault tolerance and vote dispersion tolerance.

Note that there is a tradeoff between fault tolerance and vote dispersion tolerance: when N remains the same, the larger the fault tolerance, the smaller the vote dispersion tolerance. Intuitively, in a Byzantine Broadcast problem [22] where all nodes agree on a message from a leader, there is no vote dispersion tolerance as there is a unique $\mathcal{A}$ to be agreed. On the other hand, when the votes for different options are gradually dispersed, the vote dispersion tolerance becomes higher, thus inevitably sacrificing the fault tolerance. What's more, the weights of B_G and C_G are not identical, i.e., the number of the second-highest vote can increase more on vote dispersion tolerance, whereas other options have less impact. Non-faulty nodes can therefore utilize this difference to improve the consensus success rate, e.g., when a non-faulty node is struggling between two options, vote for the third option.

B. Voting Validity: A Probability Point of View

In section VI-A, the vote dispersion threshold is proposed and regarded as a trade-off with the fault tolerance, i.e., if the system is required to tolerate more faulty nodes, it can only tolerate fewer votes to the non-highest options that the non-faulty nodes voted for. In practical application scenarios, without loss of generality, we can assume a priori of node preferences for different options, i.e., good nodes vote for different options following a specific distribution. Therefore, we can use this prior knowledge to characterize the possible vote dispersion of the system and to evaluate the fault tolerance ability of the system. In particular, the results also show that voting validity cannot always be guaranteed with Byzantine nodes as a way to demonstrate the necessity of SCT proposed in Section V.

We assume that the preferences of non-faulty nodes for different options obey a multinomial distribution, with p_i denoting the probability of a non-faulty node's votes for option i. Additionally, non-faulty nodes are assumed to be independent in voting. We suggest N_G as the number of non-faulty nodes, X_i as the random variable of the number of non-faulty nodes that support option i. Let A_G and B_G be the random variables

of the number of highest and second-highest votes from non-faulty nodes. According to the probability density function (p.d.f) of the multinomial distribution:

$$P\left(X_1 = x_1, X_2 = x_2, \ldots, X_{|\mathbb{V}|} = x_{|\mathbb{V}|}\right) = \frac{n!}{x_1! x_2! \cdots x_{|\mathbb{V}|}!} p_1^{x_1} p_2^{x_2} \cdots p_{|\mathbb{V}|}^{x_{|\mathbb{V}|}}. \tag{9}$$

Note here if $\sum_{i=1}^{|\mathbb{V}|} x_i \neq N_G$, the corresponding probability will be 0, due to the property of the multinomial distribution. The cumulative distribution function (c.d.f) of multinomial distribution can be obtained.

$$\begin{aligned} &\Pr(X_1 \leq x_1, ..., X_{|\mathbb{V}|} \leq x_{|\mathbb{V}|}) \\ &= \sum_{t_1=0}^{x_1} ... \sum_{t_{|\mathbb{V}|}=0}^{x_{|\mathbb{V}|}} \Pr(X_1 = t_1, ..., X_{|\mathbb{V}|} = t_{|\mathbb{V}|}). \end{aligned} \tag{10}$$

According to Theorem 12, we consider BFT and $K = 2$. The probability satisfies Theorem 12 is:

$$\begin{aligned} &\Pr\left(A_G - B_G > t\right) \\ &= \sum_{a_G=t+1}^{N_G} \Pr\left(A_G = a_G\right) \Pr\left(B_G \leqslant a_G - t - 1 \,|A_G = a_G\right), \end{aligned} \tag{11}$$

where $Pr(A_G = a_G)$ can be obtained as

$$\begin{aligned} \Pr\left(A_G = a_G\right) &= \Pr\left(A_G \leqslant a_G\right) - \Pr\left(A_G \leqslant a_G - 1\right) \\ &= \Pr\left(x_1 \leqslant a_G, ..., x_{|\mathbb{V}|} \leqslant a_G\right) \\ &\quad - \Pr\left(x_1 \leqslant a_{G-1}, ..., x_{|\mathbb{V}|} \leqslant a_{G-1}\right). \end{aligned} \tag{12}$$

Since the option with the most votes A_G can be all possible options, we have

$$\begin{aligned} &\Pr\left(B_G \leqslant a_G - t - 1 \,|A_G = a_G\right) \\ &= \sum_{i \in \mathbb{V}} \Pr\left(\left(X_i = a_G \cap \left(\underset{j \in \mathbb{V}/i}{\cap} X_j \leqslant a_G - t - 1\right)\right)\right) \\ &= \sum_{i \in \mathbb{V}} \Pr(X_1 \leqslant a_G - t - 1, ..., X_i \leqslant a_G, ..., X_{|\mathbb{V}|} \leqslant a_G - t - 1) \\ &- \Pr\left(X_1 \leqslant a_G - t - 1, ..., X_i \leqslant a_G - 1, ..., X_{|\mathbb{V}|} \leqslant a_G - t - 1\right). \end{aligned} \tag{13}$$

Based on p.d.f and c.d.f of multinomial distribution, we yield $\Pr\left(A_G - B_G > t\right)$ combining (12) and (13).

We give a simulation result of $\Pr\left(A_G - B_G > t\right)$ given the maximum number of Byzantine nodes t under different input distributions. Particularly, we let the number of non-faulty nodes $N_G = 10$, and take four specific multinomial distribution $D_1 \sim D_4$ (shown in Figure 1(a)) as examples to demonstrate how vote dispersion weakens faulty tolerance. The node preference distribution $\{p_i\}$ shown in Figure 1(a) implies an initial system entropy H_0 (multiplied by the number of good nodes), which is shown in the legend of Figure 1. Based on Equation (11), Figure 1(b) demonstrates the probability of system tolerating different maximum numbers of faulty nodes t when guaranteeing voting validity. Note that when $t \geq 5$ given $N_G = 10$, the consensus protocol would fail since $N \leq 3t$, but it is not shown in the figure.

Additionally, Figure 1(c) shows the system entropy of achieving voting validity H_s, which is obtained by $H_s = -(1-p_v)log(1-p_v) - p_v log(p_v)$ where p_v refers to $Pr(A_G - B_G > f)$ when $f \neq 0$ and $p_v = 1$ when $f = 0$. Note here when $f = 0$, voting validity is guaranteed; thus, the probability of achieving voting validity is 1. Figure 1(c) shows that when $f = 0$, achieving voting validity is deterministic and $H_s = 0$. When Byzantine nodes are introduced, H_s increases sharply. Then H_s decreases with larger f, inclining not to achieve the voting validity deterministically. Additionally, the result indicates that for a fixed f, the greater the initial system entropy H_0 (determined by the distribution of the inputs), the greater the interference from the Byzantine nodes, i.e., smaller H_s.

The above probabilistic analysis also indicates the impossibility result of voting validity from another perspective. Meanwhile, the importance of SCT is also demonstrated, where the output is 100% guaranteed to have voting validity.

VII. Protocol Refinement

In this section, we first optimize the proposed protocol for optimistic responsiveness in Section IV-A, then implement the local broadcast model in wireless networks. The analysis shows that the former increases the efficiency of the protocol's operation under synchronous networks, while the latter leads to an increase in fault tolerance ability. Here we choose BFT and Algorithm 1 for optimization, while CFT and SCT can be easily optimized similarly to get the above results.

A. Incremental Threshold Protocol

From the perspective of protocol design, the algorithms proposed in this paper differ from other BA/BB protocols in terms of the node's local judgment condition. We focus on $A_G - B_G$ caused by multiple input values and aim to ensure $A_G - B_G$ exceeds a threshold. In contrast, other BA/BB protocols focus on only one option and track whether the number of messages reaches a certain limit (i.e., quorum). Once reaching this limit, a protocol can perform the next operation, referred to as **optimistic responsiveness** [34]. For example, in PBFT [12], nodes wait for $N - t$ valid messages from other nodes to enter the next phase.

Luckily, these two design approaches are not opposites and can be merged. The key to reaching optimistic responsiveness is to focus on x votes that a node has not received due to the delay δ_t in synchronous network assumption. To ensure finally $A_i - B_i > \delta_P$ and propose $\mathcal{A}_i$, one should consider the worst case that all x votes that are not yet received support local second-highest-voted option $\mathcal{B}_i$. This yields $A_i - (B_i + x) > \delta_P$, where $x = N - A_i - B_i - C_i$. So we have

$$A_i > \frac{N - C_i + \delta_P}{2}. \tag{14}$$

The above derivation implies if votes received by a non-faulty node at a time satisfy Inequality (14), the node is safe to propose $\mathcal{A}_i$. This conclusion gives us a new perspective:

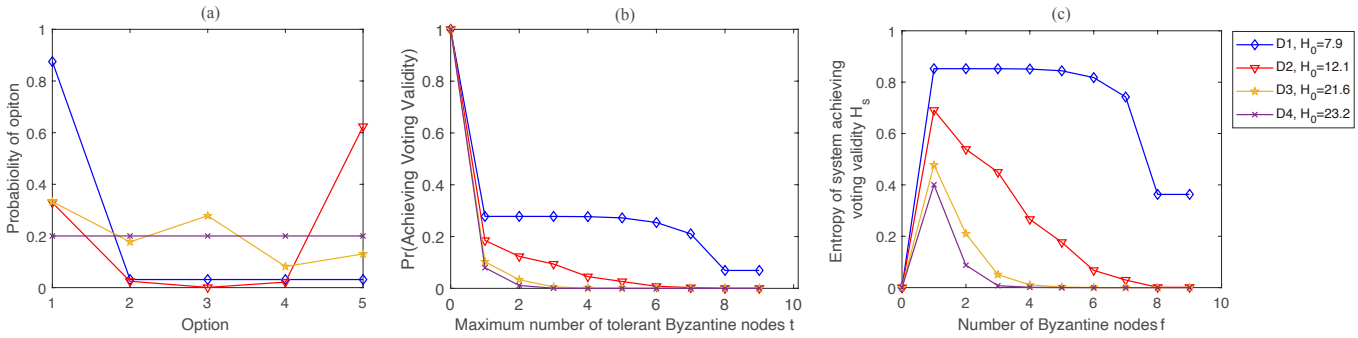

Fig. 1. Simulation from the Probability Point of View

when receiving votes from other nodes, this inequality will eventually hold as A_i and C_i are monotonically increasing. Therefore, the nodes do not need to collect all the votes before finally proposing their local maximum. This leads to "wait for $2\delta_t$ to receive all votes" in Line 11 of Algorithm 1 unnecessary, thus speeding up the operation of the protocol and increasing the efficiency. This feature also allows the protocol to be used in *partially synchronous networks*, as the protocol works regardless of what the actual δ_t is. Algorithm 3 below shows the change compared with Algorithm 1 to reach the optimistic responsiveness property, where only Phase 3 is changed. Note that Algorithm 3 is for BFT, so $\delta_P = 0$; the SCT protocol can also be easily modified using $\delta_P = t$.

Algorithm 3: BFT Voting Protocol with Incremental Threshold (Change Phase 3 in Algorithm 1)

```
▷ Phase 3: Propose Phase
/* Receive and Sort Ballots                    */
10 while receiving at least t+1 votes of s do
11     V_i ← Votes received;
12     A_i, B_i, C_i = Sort(V_i);
       /* Propose Option;For BFT, δ_P = 0      */
       /* Optimistic Responsiveness            */
13     if A_i > (N−C_i)/2 then
14         Broadcast(propose A_i);
15         Break;
```

We give an example of how such an incremental threshold differs from the original design. Assume there is a total of 10 nodes, and $\{0, 0, 1, 0, 0, 0, 2, 3, 0, 1\}$ are received successively by a non-faulty node in a protocol run. According to Inequality (14), $A_i > (N - C_i)/2$ is already satisfied when receiving the seventh "2", so the node can immediately propose "0". The incremental threshold optimizes the protocol's speed as the correct decision is made after receiving only 7 votes, rather than 10 votes.

B. Consensus in Wireless Broadcast Networks

As is well known, the inherent broadcast nature of wireless networks gives Byzantine fault-tolerant systems a higher degree of fault tolerance ability as a Byzantine node can be restricted to send different messages to different nodes [35]. This property can enhance the fault tolerance of existing protocols under wireless communication scenarios, e.g., multi-agent systems such as UAV swarms and vehicle clusters.

We propose in Algorithm 4 below a BFT protocol that achieves voting validity in a synchronous system given

$$N > 2t + 2B_G + C_G. \tag{15}$$

In Algorithm 4, the function BROADCAST implies the Local Broadcast Model as defined in Section III.

Algorithm 4: BFT Voting Protocol with Local Broadcast Model (Change Phase 1 and 2 in Algorithm 1)

```
/* For Speaker ONLY                            */
▷ Phase 1: Prepare Phase
1 Broadcast(Subject s);
/* For All Nodes                               */
▷ Phase 2: Vote Phase
/* Local Broadcast Perferences                 */
2 if receiving subject s then
3     Node Preference: v_i;
4     Broadcast(m, i → v_i);
```

Property 6. *[36] Assume there are two subjects s and s'. All non-faulty nodes will not agree on both s and s' given $N > 2t$ with Local Broadcast Model and complete communication graph.*

Proof. The property is a degraded version of the conclusion in [36]. Here we give simple proof from the quorum point of view. By means of contradiction, assume both s and s' are agreed, and there are $N - t$ votes from the distributed system that support s and s' respectively, to ensure the termination property. So there are in total of $2(N - t) - N = N - 2t > 0$ nodes who vote for both s and s'. However, this is impossible with the Local Broadcast Model, as a node cannot send different messages to different nodes. Thus, this assumption leads to a contradiction. □

The proof of correctness is analog to Theorem 9 but should consider Property 6, and we omit the proof here. Compared to Algorithm 1, the Byzantine Broadcast in Line 6 of Algorithm 1 is no longer needed. In addition, the threshold in Algorithm 4 no longer requires $N > 3t$, thus increasing the protocol's fault tolerance ability.

VIII. Conclusion

In this paper, we considered the multi-valued consensus problem with exactness. We define voting validity as the output being the exact plurality of non-faulty inputs and present the tolerance lower bound to achieve it. This study also indicates several impossibility results and gives practical BFT consensus protocols that satisfy the voting validity. Our result provides a new perspective of tolerance which distinguishes fault tolerance and vote dispersion tolerance. In addition, two protocol refinement methods are introduced to lower the threshold and improve the efficiency.

Further work can be done to discover the exactness of the output in other variants of validity, and explore voting validity to multi-dimensional consensus [25]. From the application perspective, the proposed protocols are single-shot consensus algorithms, thus not yet directly applicable in some distributed scenarios. Therefore, future designs can focus on designing more resilient and efficient protocols for voting in faulty distributed systems.

References

[1] Roger Wattenhofer. *The science of the blockchain*. Inverted Forest Publishing, 2016.

[2] Yang Xiao, Ning Zhang, Wenjing Lou, and Y Thomas Hou. A survey of distributed consensus protocols for blockchain networks. *IEEE Communications Surveys & Tutorials*, 22(2):1432–1465, 2020.

[3] Gil Neiger. Distributed consensus revisited. *Information processing letters*, 49(4):195–201, 1994.

[4] Russell Turpin and Brian A Coan. Extending binary Byzantine agreement to multivalued Byzantine agreement. *Information Processing Letters*, 18(2):73–76, 1984.

[5] David Stolz and Roger Wattenhofer. Byzantine agreement with median validity. In *Proceedings of the 19th International Conference on Principles of Distributed Systems (OPODIS 2015)*, volume 46, page 22. Schloss Dagstuhl–Leibniz-Zentrum für Informatik GmbH, 2016.

[6] Darya Melnyk and Roger Wattenhofer. Byzantine agreement with interval validity. In *Proceedings of the 2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*, pages 251–260, 2018.

[7] Hector Garcia-Molina. Elections in a distributed computing system. *IEEE transactions on Computers*, 31(01):48–59, 1982.

[8] Davide Grossi. Social choice around the block: On the computational social choice of blockchain. *arXiv preprint arXiv:2203.07777*, 2022.

[9] Xiangke Wang, Zhiwen Zeng, and Yirui Cong. Multi-agent distributed coordination control: Developments and directions via graph viewpoint. *Neurocomputing*, 199:204–218, July 2016.

[10] Chenglin Feng, Zhangchen Xu, Xincheng Zhu, Paulo Valente Klaine, and Lei Zhang. Wireless distributed consensus in vehicle to vehicle networks for autonomous driving. 07 2021.

[11] Zongyao Li, Lei Zhang, Xiaoshuai Zhang, and Muhammad Imran. Design and implementation of a raft based wireless consensus system for autonomous driving. In *Proceedings of the 2022 IEEE Global Communications Conference*, pages 3736–3741, 2022.

[12] Miguel Castro, Barbara Liskov, et al. Practical Byzantine fault tolerance. In *OsDI*, volume 99, pages 173–186, 1999.

[13] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Annual Technical Conference (Usenix ATC 14)*, pages 305–319, 2014.

[14] Maher Alharby and Aad Van Moorsel. Blocksim: a simulation framework for blockchain systems. *ACM SIGMETRICS Performance Evaluation Review*, 46(3):135–138, 2019.

[15] Shehar Bano, Mustafa Al-Bassam, and George Danezis. The road to scalable blockchain designs. *USENIX; login: magazine*, 42(4):31–36, 2017.

[16] Behrooz Parhami. Optimal algorithms for exact, inexact, and approval voting. In *FTCS*, pages 404–411, 1992.

[17] Soma Chaudhuri. More choices allow more faults: Set consensus problems in totally asynchronous systems. *Information and Computation*, 105(1):132–158, 1993.

[18] Danny Dolev, Nancy A Lynch, Shlomit S Pinter, Eugene W Stark, and William E Weihl. Reaching approximate agreement in the presence of faults. *Journal of the ACM (JACM)*, 33(3):499–516, 1986.

[19] John H Wensley, Leslie Lamport, Jack Goldberg, Milton W Green, Karl N Levitt, Po Mo Melliar-Smith, Robert E Shostak, and Charles B Weinstock. Sift: Design and analysis of a fault-tolerant computer for aircraft control. *Proceedings of the IEEE*, 66(10):1240–1255, 1978.

[20] John C Knight. Safety critical systems: challenges and directions. In *Proceedings of the 24th international conference on software engineering*, pages 547–550, 2002.

[21] M. Pease, R. Shostak, and L. Lamport. Reaching Agreement in the Presence of Faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[22] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. In *Concurrency: the Works of Leslie Lamport*, pages 203–226. 2019.

[23] Matthias Fitzi and Juan A Garay. Efficient player-optimal protocols for strong and differential consensus. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 211–220, 2003.

[24] Silvia Bonomi, Antonella Del Pozzo, Maria Potop-Butucaru, and Sébastien Tixeuil. Approximate agreement under mobile Byzantine faults. *Theoretical Computer Science*, 758:17–29, 2019.

[25] Hammurabi Mendes, Maurice Herlihy, Nitin Vaidya, and Vijay K Garg. Multidimensional agreement in Byzantine systems. *Distributed Computing*, 28(6):423–441, 2015.

[26] J de V Graaff. *Theoretical welfare economics*. Cambridge University Press, 1968.

[27] Himanshu Chauhan and Vijay K Garg. Democratic elections in faulty distributed systems. In *Proceedings of the International Conference on Distributed Computing and Networking*, pages 176–191. Springer, 2013.

[28] Darya Melnyk, Yuyi Wang, and Roger Wattenhofer. Byzantine preferential voting. In *Proceedings of the International Conference on Web and Internet Economics*, pages 327–340. Springer, 2018.

[29] Lewis Tseng. Voting in the presence of Byzantine faults. In *Proceedings of the 2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 1–10. IEEE, 2017.

[30] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[31] Chiu-Yuen Koo. Broadcast in radio networks tolerating Byzantine adversarial behavior. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 275–282, 2004.

[32] Kartik Nayak, Ling Ren, Elaine Shi, Nitin H Vaidya, and Zhuolun Xiang. Improved extension protocols for Byzantine broadcast and agreement. *arXiv preprint arXiv:2002.11321*, 2020.

[33] Rachid Guerraoui, Vassos Hadzilacos, Petr Kuznetsov, and Sam Toueg. The weakest failure detectors to solve quittable consensus and nonblocking atomic commit. *SIAM Journal on Computing*, 41(6):1343–1379, 2012.

[34] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–33. Springer, 2018.

[35] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 73–85, 1989.

[36] Muhammad Samir Khan, Syed Shalan Naqvi, and Nitin H Vaidya. Exact Byzantine consensus on undirected graphs under local broadcast model. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 327–336, 2019.