THE NATIONAL UNIVERSITY
of SINGAPORE

School of Computing
Computing 1, 13 Computing Drive, Singapore 117417

TRA9/23

Using Multi-dimensional Quorums for Optimal Resilience in Multi-resource Blockchains

Yucheng Sun, Ruomu Hao and Haifeng Yu

September 2023

Technical Report

Foreword

This technical report contains a research paper, development or tutorial article, which has been submitted for publication in a journal or for consideration by the commissioning organization. The report represents the ideas of its author, and should not be taken as the official views of the School or the University. Any discussion of the content of the report should be sent to the author, at the address shown on the cover.

TAN Kian Lee
Dean of School

Using Multi-dimensional Quorums for Optimal Resilience in Multi-resource Blockchains

(Technical Report)

Yucheng Sun
Department of Computer Science
National University of Singapore
Republic of Singapore
suny@u.nus.edu

Ruomu Hou
Department of Computer Science
National University of Singapore
Republic of Singapore
houruomu@comp.nus.edu.sg

Haifeng Yu
Department of Computer Science
National University of Singapore
Republic of Singapore
haifeng@comp.nus.edu.sg

***Abstract*—Permissionless blockchains commonly use resource challenges to defend against sybil attacks. For example, popular resource challenge designs include Proof-of-Work and Proof-of-Stake. It is well-known that simultaneously exploiting multiple resources can help make a permissionless blockchain more robust. Existing efforts along this direction, however, all fail to provide a complete answer to the central question of *how to combine PoW and PoS, or multiple resources in general, to achieve optimal resilience*. As our central contribution, this work gives complete answers to this central question.**

I. Introduction

A. Background

Resource challenges. In *permissionless* blockchains, an adversary can freely generate unlimited number of fake identities/nodes to attack the blockchain. All permissionless blockchains, fundamentally, use a *resource challenge* mechanism to properly defend against such an adversary. With resource challenges, the amount of adversarial influence will be proportional to the amount of resource controlled by the adversary, despite the potentially unlimited number fake identities.

Proof-of-Work (PoW) [1]–[6], which challenges computational resource, is a most widely-used design of resource challenge. With PoW, nodes in system try to solve *PoW puzzles*. Whoever solves the puzzle will be allowed to for example, propose a new block, or join a committee that runs a consensus protocol to choose the next block. Proof-of-Stake (PoS) [7]–[9] is another highly popular design, where a node's influence on the blockchain is proportional to the amount of stake (e.g., cryptocurrency) that it holds. There are also other types of resource challenges such as Proof-of-Space [10]–[12].

Combing multiple resources. It is well-known [13]–[18] that simultaneously exploiting multiple resources can make a permissionless blockchain more robust. For example, a PoW blockchain can be broken by an adversary controlling more than $\frac{1}{2}$ of the computation power in the system — hence the well-known "51% attack" on Bitcoin. But if we combine PoW and PoS, then to break the system, the adversary may need to further control some stake (in addition to the majority computational power).

Combining multiple resources can also help facilitate bootstrapping [16]. For example, at its initial stage, a blockchain may have limited total computational power. For better robustness, we can give vast majority of the weight to PoS, at this initial stage. Later, the weight of PoS can be decreased, and be re-allocated to PoW.

This research is partly supported by the Ministry of Education, Singapore, under its MOE Academic Research Fund Tier 2 (research grant number: MOE-T2EP20221-0002). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the Ministry of Education, Singapore.

B. Existing Results

Security region. To understand the resilience achieved by combining PoW and PoS, it will be convenient to consider the 2D coordinate system in Figure 1. Here, a *point* (w, s) in the coordinate system corresponds to an adversary that controls w fraction of the total computational power and s fraction of the total stake in the system. We call a (potentially infinite) set Υ of points as a *security region* or simply *region*, if Υ satisfies the following property: If Υ contains a point (w, s), then Υ must contain (w', s') for all $0 \leq w' \leq w$ and $0 \leq s' \leq s$. Roughly speaking, if a blockchain can tolerate all the points (i.e., adversaries) in a region (except the boundary), then we say that it *tolerates* that region. Section II will give a formal definition.

Triangular region and PoW difficulty. There have been a number of prior efforts on combining PoW and PoS [13]–[19] to achieve better resilience. The first attempts (e.g., [13] and [14]), unfortunately, do not come with formal security analysis.

The first design that has formal security guarantees for combining PoW and PoS is by Duong et al. [15], which is further generalized to multi-resources in [17]. Based on the results in [18], roughly speaking, their design [15] can at most tolerate the *triangular region* as in Figure 1(a).

Duong et al.'s design [15] assumes that the total amount of computational power in the system is *fixed*. To avoid this impractical assumption, Chepurnoy et al. [18] extends the design, using adaptive adjustment of *PoW difficulty* to normalize

the computational power. Here *PoW difficulty* corresponds to the amount of computation needed to solve a PoW puzzle.

Trapezoidal regions and PoW difficulty. Recently in CCS 2022, Fitzi et al. [16] propose a new design for combining multiple resources, which can tolerate all *trapezoidal regions* as in Figure 1(b) and 1(c). More precisely, their blockchain allows different *configurations*. For any given trapezoidal region, they can *configure* their blockchain to properly tolerate that region. Note that the blockchain cannot *simultaneously* tolerate two different trapezoidal regions, which is fundamentally impossible [16]. This is also why different configurations are needed. More generally, Fitzi et al. [16] have proved that a blockchain can never simultaneous tolerate (w, s) and $(1-w, 1-s)$, as illustrated in Figure 1(f).

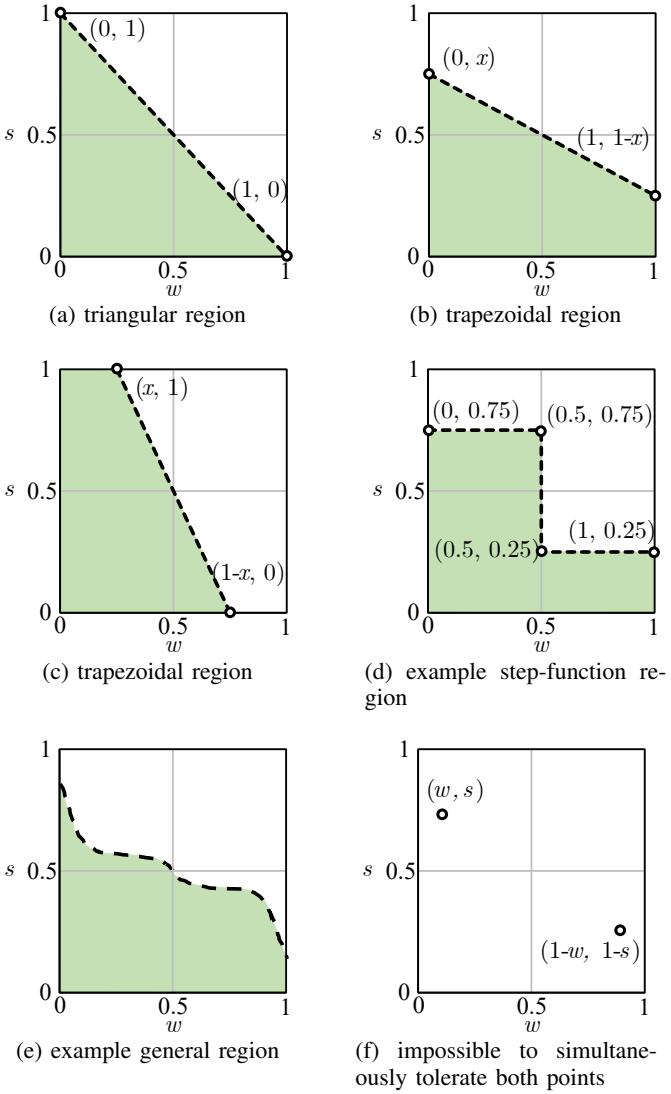
Fitzi et al. [16] also face the challenge of adjusting PoW difficulty. Specifically, Fitzi et al.'s design aims for *fungibility* between PoW and PoS — conceptually, one unit of computation power is viewed as equivalent to some units of stake. For example, to tolerate the triangular region in Figure 1(a) (which is a special case of trapezoidal region), one unit of computation power should conceptually map to $\frac{s_0}{w_0}$ units of stakes. Here w_0 and s_0 are the total units of computational power and stake in the system, respectively. As w_0 increases/decreases, conceptually this *exchange rate* of $\frac{s_0}{w_0}$ should decrease/increase accordingly. To do so, Fitzi et al. [16] adjust the PoW difficulty based on past observations on w_0. As a feedback-based adaptive process, fundamentally, their design critically relies on the fluctuation of w_0 being well-bounded.

C. The Central Question Is Still Open

Despite all these existing efforts, the central research question still remains open, regarding *how to combine PoW and PoS (or multiple resources in general) to achieve optimal resilience*. Specifically:

First, since it is impossible [16] for a blockchain to simultaneous tolerate (w, s) and $(1-w, 1-s)$, let us call a region Υ as *self-contradicting*, if for some $0 \le w \le 1$ and $0 \le s \le 1$, the region Υ contains both (w, s) and $(1-w, 1-s)$. Can we design a protocol with *optimal resilience*? Here we define *optimal resilience* as being able to tolerate every non-self-contradicting region. Existing designs can only tolerate trapezoidal regions. But there are many other regions (e.g., Figure 1(d) and 1(e))) that are not self-contradicting. Can we tolerate those? In fact, Fitzi et al. [16] explicitly raise this question as future work.

Answering this question provides a *complete characterization* on the resilience offered by combining PoW and PoS. The answer also bears practical importance: Consider a simple scenario with two major mining pools, each controlling 49% of the computational power in the system. Imagine that we are primarily concerned with the potential compromise of each mining pool (instead of individual nodes), and that the remaining 2% of the computational power is well-protected. Then the adversary controls either 0%, 49%, or 98% of the computational power. If the blockchain can tolerate the region in Figure 1(d), then it can tolerate s close to $0.75/0.75/0.25$, respectively, when $w = 0/0.49/0.98$. Such tolerance is strictly better than any trapezoidal region.



(a) triangular region (b) trapezoidal region (c) trapezoidal region (d) example step-function region (e) example general region (f) impossible to simultaneously tolerate both points

Fig. 1: Security regions.

Second, adjusting PoW difficulty is a common intricacy [16], [18] when combining PoW and PoS. When adjusting PoW difficulty, Chepurnoy et al. [18] do not offer formal security analysis, while Fitzi et al. [16] require a fundamental assumption that w_0 does not fluctuate too much.[1] Failing to properly adjust PoW difficulty will prevent the blockchain from tolerating the region that it is configured to tolerate. Hence these existing solutions are not ideal.

D. Our Work and Contribution

As our central contribution, we propose the novel design and formal security analysis of a blockchain protocol that combines PoS and PoW, which can be further generalized to multiple resources, with the following key features:

[1]This assumption has a further complication: More fluctuation forces their design to use longer epochs. But with longer epochs, the fluctuation (from epoch to epoch) may further increase. Hence this may not even converge.

- Our blockchain is the *very first* blockchain for combining multiple resources that has *optimal resilience*. Namely, our formal security analysis proves that for every possible security region Υ, as long as Υ is not self-contradicting, our blockchain can be configured to tolerate Υ. Among others, our blockchain can tolerate the regions in Figure 1(d) and 1(e). *No prior blockchain designs can tolerate such regions.*
- Our blockchain completely avoids the need for PoW difficulty adjustment. In fact, we do not even need the notion of PoW difficulty.
- We do not sacrifice other aspects, and we do not need additional assumptions in the security model.

We have further implemented a research prototype of our blockchain design. Our experiments show that our prototype can deliver good end-to-end performance to the end users, in terms of both transaction throughput and block confirmation latency.

E. Our Key Techniques

Existing designs [15], [16], [18] for combining PoW and PoS typically use the following high-level design: Nodes holding computational resources will generate PoW blocks, while nodes holding stakes will generate PoS blocks. The protocol then maintains strategically-designed relations/pointers among all these PoW blocks and PoS blocks, to enable the blockchain to tolerate the target security region. Finally, a longest-chain rule (or some variant of it) is applied to all the blocks to get the "main chain".

Our blockchain instead uses a byzantine agreement (BA) protocol as its core. We do not have separate PoW blocks and PoS blocks, and there is only one kind of blocks in our blockchain. To tolerate every non-self-contradicting region:

First, we avoid combining PoW and PoS into one measure, and avoid having any exchange rate. But we still let both PoW and PoS influence the BA protocol. In a typical BA protocol, nodes exert influence by casting votes at various steps. For example in certain step, a certain value may be adopted by the BA protocol, if a *majority quorum* of nodes have voted for that value.

As our central idea, which is simple yet elegant, we introduce the novel concept of *2D-quorums*. With 2D-quorums, we do not determine the quorum based on the number of votes. Instead, we separately consider the set G_w of PoW votes, and the set G_s of PoS votes. We then design an appropriate *mapping*, to map this tuple $\langle G_w, G_s \rangle$ to a certain point z on the coordinate system in Figure 1. (These mappings will be integral part of the 2D-quorum design.) At the same time, we use the security region Υ itself, as the configuration of our blockchain. We define a *2D-quorum* as being met, if $z \in \Upsilon$. We note that our novel concept of 2D-quorums could also be of separate independent interest, beyond blockchains.

Second, we do not set any PoW difficulty – instead, each node simply keeps doing hashes, and remembers the smallest hash values that it has got. Next, we use *in-network aggregation* to find out the m globally minimum hash values. One subtlety here, however, is that without having an appropriate PoW difficulty, these m hash values only give nodes low-quality information on how to construct the mapping from $\langle G_w, G_s \rangle$ to z, as needed by the 2D-quorum design. We will carefully deal with this.

II. System Model and Attack Model

We consider a standard permissionless setting without PKI. Each node in system has a *locally-generated* public/private key pair. The public key is also used as the *id* of the node. A node is either *honest* or *malicious*. An honest node follows the protocol faithfully, while a malicious node may deviate from the protocol arbitrarily. Hence a malicious node is fully byzantine. We assume that all malicious nodes are controlled by some *adversary*, and they can collude in arbitrary ways. We make no assumption on the number of malicious nodes.

We assume that there are two types of resources in the system, *computation* and *stake*. (We generalize to more resources later.) We use f'_w (f'_s) to denote the fraction of computational power (stake), out of all computational power (stake) in the system, that is controlled by the adversary. We say that a blockchain *tolerates* a point (w, s) in Figure 1, if it is robust against the adversary with $f'_w = w$ and $f'_s = s$.

To facilitate discussion, we view each honest node as holding either some computational power (in which case the node is a *PoW node*) or some stake (in which case the node is a *PoS node*), but not both. It is trivial to generalize: If a node does hold both computational power and stake, then it will double-count as both a PoW node and a PoS node.

We say that a blockchain (under a certain configuration) *tolerate* a security region Υ, if the blockchain can tolerate every point (w, s) where $(w + \epsilon, s + \epsilon) \in \Upsilon$. Here ϵ is some small positive constant (e.g., 0.001) that is given a prior. This ϵ term is needed due to a technicality: Large-scale blockchains typically use sampling to reduce overhead, and this ϵ term absorbs the sampling error.[2]

To achieve optimal resilience, our blockchain design will consider any given security region Υ that is not self-contradicting, and any adversary such that $(f'_w+\epsilon, f'_s+\epsilon) \in \Upsilon$. We further define $f_w = f'_w + \epsilon$ and $f_s = f'_s + \epsilon$, which implies $(f_w, f_s) \in \Upsilon$.

As in typical large-scale blockchains [2], [4], [6], [9], [20], we assume that all the nodes form an overlay network, and that the subgraph containing all the honest nodes is connected. We use d to denote an upper bound on the diameter of this subgraph. We assume that there is a known upper bound on message propagation delay, which eventually enables the use of rounds in the protocol. We model hash functions as random oracles. We focus on consensus-based blockchains, which is by far the most common type of blockchains.[3] We assume

[2]Prior works also need this technicality: For example, when we say that Fitzi et al.'s design [16] tolerates trapezoidal regions, there is also an implicit ϵ term.

[3]It is also possible to design blockchains based on byzantine broadcast (e.g, [21]), but those blockchains are still quite uncommon today due to their relatively high performance overheads.

the adversary is slowly-adaptive, in the sense that it takes multiple epochs for the adversary to corrupt nodes. Note that all these assumptions are the same as in the state-of-the-art approach [16] of combining PoW and PoS.

We say that a probability is *negligible* if it is on the order of $e^{-\Omega(\min(m,k))}$, where m and k are (security) parameters in our protocol.

III. Overview of Our Design and Security Analysis

Design overview. In our blockchain, at fixed time intervals (e.g., 3 minutes), all nodes invoke a byzantine agreement protocol `Generalized_BA()` that we have designed, to generate the next block in the blockchain. The security of the blockchain all comes from the guarantees of `Generalized_BA()`. Specifically, `Generalized_BA()` will guarantee that i) **[agreement]** `Generalized_BA()` returns the same block on all honest nodes; and ii) **[validity]** if the block proposer is honest, then `Generalized_BA()` must return the block proposed by the block proposer. There are no forks in our blockchain.

Internally, `Generalized_BA()` extensively uses 2D-quorums. To tolerate any given security region Υ (that is not self-contradicting), we use Υ itself as the configuration for our blockchain. Our design then internally uses Υ in the definition of 2D-quorums, which in turn enabled `Generalized_BA()` to tolerate adversaries in Υ.

Security analysis overview. Our security analysis will interleave with the discussions of our design in the remainder of this paper. To facilitate understanding, here we provide a top-down overview of the security proofs.

To prove the security of our blockchain, it suffices to show that `Generalized_BA()` can guarantee **[agreement]** and **[validity]** as explained earlier. In `Generalized_BA()`, to reduce overhead, a subset of the nodes (called *core nodes*) will invoke another subroutine called `Generalized_BA_Core()`. Theorem 6 will prove that the return values of `Generalized_BA_Core()` on these core nodes must satisfy **[agreement]** and **[validity]**. After getting its return value, a core node will vote for that value, and disseminate the vote (with signature) to all other nodes. A non-core node will adopt a value (which corresponds to a block) y, if the set of votes on y constitute a 2D-quorum.

Next, we design `Generalized_BA_Core()` by adapting an existing BA protocol in [22]–[24]. The protocol in [22]–[24] was originally designed only for PoS, and can tolerate adversaries holding less than 50% of the stake in the system. There is no PoW in their setting. In their protocol, there are various places that check whether a certain set G of nodes constitute a simple (1D) majority quorum. *Our key adaptation is that instead, we check whether G is a 2D-quorum.* The specific designs of the 2D-quorums in different steps of the protocol will be different. We will prove the properties of these 2D-quorums, as well as the properties of `Generalized_BA_Core()` when using these 2D-quorums. All these eventually enable us to prove Theorem 6, which captures the end-to-end guarantees of `Generalized_BA_Core()`.

Roadmap for Section IV to VI. In the next, Section IV presents our 2D-quorum design, while assuming perfect PoW difficulty adjustment. Section V removes that assumption. Section VI presents our designs of `Generalized_BA()` and `Generalized_BA_Core()`, which use 2D-quorums.

IV. 2D-Quorums — Assuming PoW Difficulty Adjustment

This section presents our 2D-quorum design, while assuming perfect PoW difficulty adjustment. To use 2D-quorums, we need each node A to first construct a *PoS committee* ($\mathbb{S}_A$) and a *PoW committee* ($\mathbb{W}_A$). Recall that in order to generate the next block on the blockchain, all nodes will invoke `Generalized_BA()`. Note that $\mathbb{S}_A$ and $\mathbb{W}_A$ are with respect to a give invocation of `Generalized_BA()`, and they may change across invocations.

A. PoS Committee

Our way of constructing the PoS committee is similar to [7]–[9]. Let t_3 be the time when the given invocation of `Generalized_BA()` starts. We first use existing *beacon generation mechanism* [7]–[9] for blockchains, to generate an unpredictable (potentially biased) random *beacon* at time t_2 where $t_2 < t_3$. This beacon will be seen by all honest node by time t_3. The amount of bias in the random beacon can be properly bounded using various techniques [7]–[9].

Next let $\mathcal{D}$ be the stake distribution (i.e., which node holds which stake) at time t_1, where $t_1 < t_2$. Here t_1 needs to be before t_2, so that the adversary cannot change $\mathcal{D}$ (by doing transactions), after the beacon has been disclosed. Note that $\mathcal{D}$ is part of the blockchain state at time t_1, and all honest nodes know $\mathcal{D}$.

Finally, a node A constructs $\mathbb{S}_A$ by choosing m uniformly random stakes (out of all stakes in $\mathcal{D}$), while using the beacon as randomness. Those nodes holding these m stakes will then be in $\mathbb{S}_A$. Note that $\mathbb{S}_A$ must be the same across all honest A's. It is possible that a node B appears multiple times in $\mathbb{S}_A$. This can be easily dealt with by giving B a weight that equals its multiplicity. To simplify discussion, however, we will assume that each B appears at most once in $\mathbb{S}_A$. We use $\mathbb{S}$ to denote the honest nodes in the PoS committee $\mathbb{S}_A$.

B. PoW Committee — Assuming PoW Difficulty Adjustment

This section explains how to construct the PoW committee $\mathbb{W}_A$, under the assumption that the PoW difficulty is perfectly adjusted, so that the entire system can generate total m PoW solutions, for some fixed m value.

Again, let t_3 be the time when the given invocation of `Generalized_BA()` starts. As earlier, we again generate an unpredictable random `beacon` at time t_0 where $t_0 < t_3$. All honest nodes will see this beacon at time t_1 that is slightly after t_0. The nodes will then solve the PoW puzzle from time t_1 to t_2, and will propagate the PoW solutions from time t_2 to t_3, where $t_1 < t_2 < t_3$, as following.

PoW puzzle solving from t_1 to t_2. A node A keeps computing `hash(beacon`$|A$'s public key$|$`nonce)`, while

TABLE I: Our key notations.

Υ	security region
d	upper bound on network diameter
m	committee size
f'_w	fraction of computational power controlled by adversary
f'_s	fraction of stake controlled by adversary
f_w	$f_w = f'_w + \epsilon$
f_s	$f_s = f'_s + \epsilon$ (By our model, we have $(f_w, f_s) \in \Upsilon$.)
$\mathbb{W}$	the set of honest nodes whose PoW solutions are all among top-m solutions in the system
$\mathbb{S}$	the set of honest nodes in the PoS committee
$\mathbb{W}_A$ and $\mathbb{S}_A$	A's view of the PoW committee and PoS committee
$\mathbb{T}^A_w$ and $\mathbb{T}^A_s$	A's view of the parameters used in 2D-quorums

trying different `nonce` values. Node A successfully solves the PoW puzzle, if it finds a `nonce` such that `hash(beacon`$|A$'s public key$|$`nonce)` $\leq$ `threshold`. The tuple $\langle A$'s public key, `nonce`$\rangle$ is called a *solution* for the PoW puzzle. Here the `threshold` parameter determines the difficulty level of the PoW puzzle. Adjusting `threshold` serves to ensure that there will be total m PoW solutions.

Propagating solutions from t_2 to t_3. At time t_2, if a node A has found a solution, it will *propagate* its solution to all nodes. Specifically, A sends its solution all its neighbors in the overlay network. Whoever receives such a solution will recursively forward the solution to its own neighbors.

Constructing PoW committee. At time t_3, each node A will have seen a set of PoW solutions. A's PoW committee $\mathbb{W}_A$ will simply be the set of public keys in these solutions. Note that $\mathbb{W}_A$'s can be different for different A's: For example, a malicious node may choose to present its PoW solution to A_1 (immediately before time t_3) but not to A_2, causing $\mathbb{W}_{A_1} \neq \mathbb{W}_{A_2}$. For the same reason, we always have $|\mathbb{W}_A| \leq m$, but not necessarily $|\mathbb{W}_A| = m$.

Let $\mathbb{W}$ be those honest nodes with PoW solutions. We must have $\mathbb{W} \subseteq \mathbb{W}_A$, since honest nodes never suppress their solutions. Furthermore, for all honest A, the set $\mathbb{W}_A$ must be a subset of (the public keys in) those m PoW solutions system-wide. Hence $|\cup_{A \text{ being honest}} \mathbb{W}_A| \leq m$.

C. Definition of 2D-Quorums

Definition. We can now define our novel *2D-quorums*: Let Υ be any given security region. For any node A, let $\mathbb{T}^A_w$ be some set of PoW nodes chosen by A, and $\mathbb{T}^A_s$ be some set of PoS nodes chosen by A, where $|\mathbb{T}^A_w| \leq m$ and $|\mathbb{T}^A_s| \leq m$. We say that a set G of nodes is a *2D-quorum* for $\langle \mathbb{T}^A_w, \mathbb{T}^A_s \rangle$ if:

$$\left(1 - \frac{|G \cap \mathbb{T}^A_w|}{m}, 1 - \frac{|G \cap \mathbb{T}^A_s|}{m}\right) \quad \in \quad \Upsilon \qquad (1)$$

Table I summarizes our key notations.

Understanding the definition. Here the set $\mathbb{T}^A_w$ can be different for different A's, since $\mathbb{T}^A_w$ will be constructed by A based on its local view/information. Furthermore, even for a given A, it may use different $\mathbb{T}^A_w$'s at different steps of our protocol. Hence we do not have a fixed $\mathbb{T}^A_w$.

The simplest use of 2D-quorums would be for A to set $\mathbb{T}^A_w = \mathbb{W}_A$ and $\mathbb{T}^A_s = \mathbb{S}_A$. The term $|G \cap \mathbb{W}_A|$ is then the number of nodes in A's PoW committee that belong to G. A simple majority quorum of the PoW committee would require $\frac{|G \cap \mathbb{W}_A|}{m} > \frac{1}{2}$ or $1 - \frac{|G \cap \mathbb{W}_A|}{m} \leq \frac{1}{2}$. As an example of 2D-quorum (instead of basic majority quorum), let us consider the Υ in Figure 1(a). Then for G to be a 2D-quorum for $\langle \mathbb{W}_A, \mathbb{S}_A \rangle$ under such Υ, Equation 1 essentially requires $|G \cap \mathbb{W}_A| + |G \cap \mathbb{S}_A| > m$. This is as expected, since it means that G contains more than m members from the two committees combined. (The two committees collectively have at most $2m$ members.)

D. Properties of 2D-Quorums

Desired properties. Ideally, we want to construct $\mathbb{T}^A_w$ and $\mathbb{T}^A_s$ for each honest node A, in such a way that the following 4 properties hold. These 4 properties are common properties needed by typical BA protocols. With these properties, one will be able to easily adapt most existing BA protocols, to make them work with 2D-quorums. The resulting BA protocol can then tolerate all adversary in the given security region Υ. Hence achieving these properties will be the key.

Definition 1: (**Desired properties from 2D-quorums**.) For every honest node A:

- **[P1]** *If $G = \mathbb{W} \cup \mathbb{S}$, then G is a 2D-quorum for $\langle \mathbb{T}^A_w, \mathbb{T}^A_s \rangle$.* Intuitively, this means that honest nodes can form a 2D-quorum.
- **[P2]** *A 2D-quorum for $\langle \mathbb{T}^A_w, \mathbb{T}^A_s \rangle$ must contain at least one (honest) node from $\mathbb{W} \cup \mathbb{S}$.* Intuitively, this means that malicious nodes by themselves cannot form a 2D-quorum.
- **[P3]** *If G is a 2D-quorum for $\langle \mathbb{T}^A_w, \mathbb{T}^A_s \rangle$, and if G' is a 2D-quorum for $\langle \mathbb{T}^B_w, \mathbb{T}^B_s \rangle$ for any honest node B, then $G \cap G' \neq \emptyset$.* Intuitively, this means that two 2D-quorums (even on two different nodes) must intersect.
- **[P4]** *Given a set G of nodes, there exist a verification mechanism for each honest node C, such that:*
  - *If G is a 2D-quorum for $\langle \mathbb{T}^A_w, \mathbb{T}^A_s \rangle$, then G must pass such verification.*
  - *If G passes such verification, then* **[P2]** *and* **[P3]** *must hold for G — namely, i) G must contain some node in $\mathbb{W} \cup \mathbb{S}$, and ii) $G \cap G' \neq \emptyset$ where G' is any 2D-quorum for $\langle \mathbb{T}^B_w, \mathbb{T}^B_s \rangle$ for any honest node B.*

  Intuitively, this means that a 2D-quorum for node A can verified by another node C.

Achieving these properties. Recall that the adversary controls f'_w fraction of the total computational power and f'_s fraction of the total stake in the system, and we have the condition that $(f'_w + \epsilon, f'_s + \epsilon) \in \Upsilon$. Also recall $f_w = f'_w + \epsilon$ and $f_s = f'_s + \epsilon$. We can now prove that simply setting $\mathbb{T}^A_w = \mathbb{W}_A$ and $\mathbb{T}^A_s = \mathbb{S}_A$ will guarantee all the above properties:

Theorem 2: (**[P1] through [P4] for $\langle \mathbb{W}_A, \mathbb{S}_A \rangle$, assuming perfect PoW difficulty adjustment.**) Assume that the PoW

difficulty is perfectly adjusted. For each honest node A, let $\mathbb{T}_w^A = \mathbb{W}_A$ and $\mathbb{T}_s^A = \mathbb{S}_A$. If $|\mathbb{W}| \geq (1-f_w)m$, $|\mathbb{S}| \geq (1-f_s)m$, and if the zone Υ is not self-contradicting, then all 4 properties **[P1]** through **[P4]** must hold.

Proof: First, recall that by definition, if a zone contains (w, s), then it contains all (w', s') where $0 \leq w' \leq w$ and $0 \leq s' \leq s$.

For **[P1]**: Let $G = \mathbb{W} \cup \mathbb{S}$. Then we have $1 - \frac{|G \cap \mathbb{W}_A|}{m} \leq 1 - \frac{|\mathbb{W}|}{m} \leq f_w$ and $1 - \frac{|G \cap \mathbb{S}_A|}{m} \leq f_s$. Hence given $(f_w, f_s) \in \Upsilon$, we must also have $(1 - \frac{|G \cap \mathbb{W}_A|}{m}, 1 - \frac{|G \cap \mathbb{S}_A|}{m}) \in \Upsilon$. Thus G is a 2D-quorum for $\langle \mathbb{W}_A, \mathbb{S}_A \rangle$.

For **[P2]**: Prove by contradiction — assume that G does not contain any node from $\mathbb{W} \cup \mathbb{S}$ and that G is a 2D-quorum for $\langle \mathbb{W}_A, \mathbb{S}_A \rangle$. This implies $(1 - \frac{|G \cap \mathbb{W}_A|}{m}, 1 - \frac{|G \cap \mathbb{S}_A|}{m}) \in \Upsilon$. Since $|\mathbb{W}| \geq (1-f_w)m$ and $\mathbb{W} \subseteq \mathbb{W}_A$, we know that $1 - f_w = 1 - \frac{m-(1-f_w)m}{m} \leq 1 - \frac{|G \cap \mathbb{W}_A|}{m}$. Similarly, $1 - f_s \leq 1 - \frac{|G \cap \mathbb{S}_A|}{m}$. By definition of a zone, we then must have $(1 - f_w, 1 - f_s) \in \Upsilon$. Together with the fact that $(f_w, f_s) \in \Upsilon$, the zone Υ now becomes self-contradicting.

For **[P3]**: Define $w = 1 - \frac{|G \cap \mathbb{W}_A|}{m}$ and $s = 1 - \frac{|G \cap \mathbb{S}_A|}{m}$. Similarly define w' and s' for G'. Prove by contradiction and assume $G \cap G' = \emptyset$. Then we would have $|G \cap \mathbb{W}_A| + |G' \cap \mathbb{W}_B| \leq m$. This in turn means $1 - w' \leq w$. Similarly we have $1 - s' \leq s$. Since G is a 2D-quorum, we have $(w, s) \in \Upsilon$. In turn, by definition of a zone, we have $(1 - w', 1 - s') \in \Upsilon$. But since G' is also a 2D-quorum, we also have $(w', s') \in \Upsilon$. The zone Υ now becomes self-contradicting.

For **[P4]**: Let G_w be the set of PoW nodes in G that have valid PoW solutions. Node C uses the following simple verification mechanism: G pass verification if and only if $(1 - \frac{|G_w|}{m}, 1 - \frac{|G \cap \mathbb{S}_C|}{m}) \in \Upsilon$. Now we prove that such verification offers the two desired guarantees. First, if G is a 2D-quorum for $\langle \mathbb{W}_A, \mathbb{S}_A \rangle$, then $(1 - \frac{|G \cap \mathbb{W}_A|}{m}, 1 - \frac{|G \cap \mathbb{S}_A|}{m}) \in \Upsilon$. Note that $1 - \frac{|G_w|}{m} \leq 1 - \frac{|G \cap \mathbb{W}_A|}{m}$ and $1 - \frac{|G \cap \mathbb{S}_C|}{m} = 1 - \frac{|G \cap \mathbb{S}_A|}{m}$. Hence $(1 - \frac{|G_w|}{m}, 1 - \frac{|G \cap \mathbb{S}_C|}{m})$ must also be in Υ. Second, if G passes verification, then **[P2]** and **[P3]** can be proved for G, in similar way as in our above proofs. □

Comment. Theorem 2, as well as our later lemmas and theorems, requires the conditions of $|\mathbb{W}| \geq (1-f_w)m$ and $|\mathbb{S}| \geq (1-f_s)m$. Note that $E[|\mathbb{W}|] = (1-f'_w)m = (1-f_w+\epsilon)m$ and $E[|\mathbb{S}|] = (1-f'_s)m = (1-f_s+\epsilon)m$. A simple Chernoff bound [25] will then show that except for negligible probability with respect to m, these two conditions will hold.

V. 2D-Quorums — Without PoW Difficulty Adjustment

Section IV assumed that the PoW difficulty is perfectly adjusted such that entire system can generate total m PoW solutions, for some fixed m value. We now explain how to avoid PoW difficulty adjustment.

PoW solving. Here a node A again uses different `nonce` values to compute `hash(beacon|`A's public key`|nonce)`. There is no difficulty level. For every `nonce` tried, the tuple $\langle A$'s public key, `nonce`$\rangle$ is already considered as a *solution* for the PoW puzzle. Hence A will get numerous solutions. We say that a solution has *better quality* than another, if the first solution leads to a smaller hash value. Node A only keeps the top m solutions, in terms of quality, among all its solutions.

In-network aggregation. When the PoW solving period ends, a node will *propagate* all its m solutions all nodes. We do this together with *in-network aggregation*, to avoid the prohibitive overhead of propagating $m \times n$ solutions on the overlay network. Specifically, the propagation is done for d rounds. In the first round, each node A sends its m solutions to all its neighbors. We also view A as a neighbor of itself. In each round $i \in [2, d]$, a node A first receives m solutions from each of its x neighbors. Next, among all mx solutions received, node A picks the top m solutions (in terms of quality) and sends those to its neighbors. Other solutions are discarded.

Finally, node A's PoW committee $\mathbb{W}_A$ will simply be those public keys in the top m solutions, among all the solutions that A receives in the last round. It is possible that in $\mathbb{W}_A$, there are 2 (or more) solutions from the same node B. Then we simply give B a weight of 2 (or more) in $\mathbb{W}_A$, and all our design will still work. To facilitate understanding, however, we simply assume that each node has a weight of 1.

Properties of 2D-quorums. There are subtle differences between the guarantees offered by this design and the one in Section IV. Specifically, here let y denote the number of hashes that a node can do during the PoW solving period. This leads to total yn PoW solutions in the entire system. Let $\mathbb{W}$ be those honest nodes whose PoW solutions are among the top m solutions (out of the yn solutions). It is easy to see that since solutions of nodes in $\mathbb{W}$ are never dropped during in-network aggregation, we must have $\mathbb{W} \subseteq \mathbb{W}_A$ for all honest A.

But for the union of the $\mathbb{W}_A$'s, we no longer have $|\cup_{A \text{ being honest}} \mathbb{W}_A| \leq m$. This then partially breaks the proof of Theorem 2. Specifically, the two properties **[P1]** and **[P2]** will continue to hold:

Theorem 3: (**[P1] and [P2] for** $\langle \mathbb{W}_A, \mathbb{S}_A \rangle$**.**) For each honest node A, let $\mathbb{T}_w^A = \mathbb{W}_A$ and $\mathbb{T}_s^A = \mathbb{S}_A$. If $|\mathbb{W}| \geq (1-f_w)m$, $|\mathbb{S}| \geq (1-f_s)m$, and the zone Υ is not self-contradicting, then properties **[P1]** and **[P2]** must both hold.

Proof: Same as the proof of Theorem 2. □

On the other hand, **[P3]** and **[P4]** may be violated. Figure 2 gives an example illustrating how **[P3]** can be violated.

Quick summary. To summarize, not adjusting PoW difficulty makes it trickier for 2D-quorums to guarantee **[P3]** and **[P4]**. The next section explains how we deal with this, in the context of our specific BA protocol.

VI. Our Design of `Generalized_BA()`

A. Design of `Generalized_BA()`

Recall that in our blockchain, at fixed time intervals, all nodes invoke a BA protocol `Generalized_BA()` to generate the next block in the blockchain. Algorithm 1 gives the pseudo-code for `Generalized_BA()`.

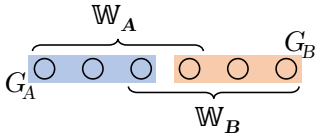


Fig. 2: Two quorums may not intersect, under a naive design. Hence $|\mathbb{W}_A| = |\mathbb{W}_B| = m = 4$ and $|\mathbb{W}_A \cup \mathbb{W}_B| > 4$. G_A (G_B) is a simple majority quorum for $\mathbb{W}_A$ ($\mathbb{W}_B$). For clarity, $\mathbb{S}_A$ and $\mathbb{S}_B$ are not shown in the figure.

Core nodes. To avoid excessive overheads, only a core set of the nodes will actually run the core byzantine agreement protocol `Generalized_BA_Core()` at Line 2 of Algorithm 1. After the core nodes get a decision from `Generalized_BA_Core()`, they will disseminate that decision value to all nodes in the overlay network. Other nodes will then adopt such decision.

Specifically, define $\mathbb{C} = \{A \mid A \in \mathbb{W}_A \cup \mathbb{S}_A\}$, and we let every node $A \in \mathbb{C}$ run `Generalized_BA_Core()`. Intuitively, such a node A believes that itself belongs to the PoW/PoS committee, according to its own local view $\mathbb{W}_A$ and $\mathbb{S}_A$. When running `Generalized_BA_Core()`, a node A will only send/receive messages to/from nodes in its local view of the PoW/PoS committee (i.e., $\mathbb{W}_A \cup \mathbb{S}_A$), which helps to reduce overheads.

Since $\mathbb{W} \subseteq \mathbb{W}_A$ and $\mathbb{S} \subseteq \mathbb{S}_A$, it is clear that all nodes in $\mathbb{W} \cup \mathbb{S}$ will run `Generalized_BA_Core()`. (Recall that by definition, all nodes in $\mathbb{W} \cup \mathbb{S}$ are honest.) Besides the nodes in $\mathbb{W} \cup \mathbb{S}$, there can be other honest nodes and malicious nodes in $\mathbb{C}$. We will ensure that `Generalized_BA_Core()` generate correct results as long as all nodes in $\mathbb{W} \cup \mathbb{S}$ run `Generalized_BA_Core()`, and regardless of the behavior of all other nodes (e.g., whether they invoke the procedure and how they may deviate).

Adopt decisions from the core nodes. A node A invoking `Generalized_BA_Core()` will get a decision value, which is the next block on the blockchain. `Generalized_BA_Core()` will ensure that (except for negligible probability) all honest nodes in $\mathbb{W} \cup \mathbb{S}$ must get the same decision. Node A will then sign and propagate its decision.

A non-core node B will adopt the decision from the core nodes, using 2D-quorums. Specifically, for any decision value y, let G_y be the set of nodes that have signed on y. Node B will adopt y if and only if G_y is a 2D-quorum for $\langle \mathbb{W}_B, \mathbb{S}_B \rangle$. By properties **[P1]** (i.e., honest nodes in $\mathbb{W} \cup \mathbb{S}$ can form a 2D-quorum) and **[P2]** (i.e., malicious nodes by themselves cannot form a 2D-quorum) in Theorem 3, we know that B must adopt the correct decision.

B. Design of `Generalized_BA_Core()`

Algorithm 2 gives the pseudo-code for `Generalized_BA_Core()`, which is invoked by the core nodes. We design `Generalized_BA_Core()` by adapting an existing BA protocol in [22]–[24]. The high-level structure of our `Generalized_BA_Core()` is the same as in [22]–[24]. Hence we do not claim novelty in this high-level structure.

High-level structure. In `Generalized_BA_Core()`, the nodes first use the beacon to select a *block proposer* for the next block on the blockchain. The selected block proposer may or may not be an honest node. Each node A then invokes `Generalized_GC_012()`, and gets a tuple $\langle \texttt{grade}_A, \texttt{block}_A \rangle$ as the return value. For different A's, this return value may be different. But we will ensure: *If some node $A \in \mathbb{W} \cup \mathbb{S}$ gets a return tuple $\langle 2, v \rangle$ for some v, then any other node $B \in \mathbb{W} \cup \mathbb{S}$ must get either $\langle 2, v \rangle$ or $\langle 1, v \rangle$. Furthermore, if the block proposer is honest, then every node in $\mathbb{W} \cup \mathbb{S}$ must get a return tuple $\langle 2, v \rangle$ for some v.*

If node A does get some return value $\langle 2, \texttt{block}_A \rangle$, then at Line 10, node A sets $x_A \leftarrow 1$, meaning that it will vote for $\texttt{block}_A$. Otherwise A sets $x_A \leftarrow 0$, to vote for the special *empty block*. This special empty block is publicly known, and contains no transactions.

Next, every node A keeps updating x_A at Line 11, for total k times, by setting $x_A \leftarrow$ `Generalized_GC_01`(x_A). If all $A \in \mathbb{W} \cup \mathbb{S}$ feed the same x_A into `Generalized_GC_01()`, then we will ensure that the return value from `Generalized_GC_01()` must be x_A. If some A's feed 0 while other A's feed 1, we will ensure that exactly one of the following three cases must occur:

- Some nodes in $\mathbb{W} \cup \mathbb{S}$ get "0" as return value from `Generalized_GC_01()`, and the remaining nodes in $\mathbb{W} \cup \mathbb{S}$ get `CommonCoin()` as return value.
- Some nodes in $\mathbb{W} \cup \mathbb{S}$ get "1" as return value from `Generalized_GC_01()`, and the remaining nodes in $\mathbb{W} \cup \mathbb{S}$ get `CommonCoin()` as return value.
- All nodes in $\mathbb{W} \cup \mathbb{S}$ get `CommonCoin()` as return value.

In the above, `CommonCoin()` is a subroutine that generates a common coin among the nodes, which we explain later. This common coin may be biased, where the amount of bias will only impact performance and not security. Let p_0 (p_1) be the probability that all honest nodes see 0 (1) as the value of the common coin. Then in the first case in the above, with probability p_0, all nodes in $\mathbb{W} \cup \mathbb{S}$ will actually get the same return value 0. Once this *anchoring event* occurs, further invocation of `Generalized_GC_01()` at Line 11 will never

Algorithm 1 `Generalized_BA()`.
Let A denote the id of invoking node.

1: **if** $A \in \mathbb{W}_A \cup \mathbb{S}_A$ **then**
2: $\quad \texttt{block}_A \leftarrow$ `Generalized_BA_Core()`;
3: $\quad$ disseminate $\langle \texttt{block}_A, \text{Sig}_A(\texttt{block}_A) \rangle$;
4: **endif**
5: for all x: $\text{SigSet}_A(x) \leftarrow \{B \mid \langle x, \text{Sig}_B(x) \rangle$ was received$\}$;
6: let y be any value such that `IsQuorum`$(\text{SigSet}_A(y), \mathbb{W}_A, \mathbb{S}_A, \Upsilon)$ holds;
7: **return** y;

Algorithm 2 `Generalized_BA_Core()`.
Let A denote the id of invoking node.

8: use `beacon` to choose a block proposer;
9: $\langle \texttt{grade}_A, \texttt{block}_A \rangle \leftarrow$ `Generalized_GC_012()`;
10: **if** $\texttt{grade}_A = 2$ **then** $x_A \leftarrow 1$ **else** $x_A \leftarrow 0$; **endif**
11: **repeat for** k **times**: $x_A \leftarrow$ `Generalized_GC_01`(x_A);
12: **if** $x_A = 1$ **then return** $\texttt{block}_A$ **else return** empty block; **endif**

change x_A to 1. Then we will have $x_A = 0$ after Line 11 on all $A \in \mathbb{W} \cup \mathbb{S}$, hence achieving agreement. Following such reasoning, one can eventually show that except for negligible probability $(1 - \min(p_0, p_1))^k$, all node $A \in \mathbb{W} \cup \mathbb{S}$ will get the same return value from `Generalized_BA_Core`().

C. Design of `Generalized_GC_01()` and `Generalized_GC_012()`

Our `Generalized_GC_01`() and `Generalized_GC_012`() subroutines are adaptations of the $\{0, 1\}$-*Graded Broadcast* and $\{0, 1, 2\}$-*Graded Broadcast* subroutines in [22]–[24]. Our main adaptation is that in various places of the protocol, instead of checking whether a set G of nodes is a majority quorum, we check whether G is a 2D-quorum.

A typical Graded Broadcast protocol simply needs the quorums to offer the four properties **[P1]** through **[P4]**, as defined in Section IV-D. As long as these properties are offered, the internal design of the quorums does not matter — for example, the quorums can be traditional majority quorums, or can be our 2D-quorums. Our Theorem 3 earlier has already proved that our 2D-quorums guarantee **[P1]** and **[P2]**. But we do not yet have **[P3]** and **[P4]**, because our PoW committee $\mathbb{W}_A$ offers only weak guarantees, due to not adjusting PoW difficulty (Section V).

To overcome this problem, we exploit the special nature of the Graded Broadcast protocol in [22]. The protocol in [22] is designed for PoS only, and considers a *sleepy model*, where some of the nodes may be offline. Their protocol then selects a committee, among the nodes that are still online/awake. To do so, conceptually, each node uses its id as the seed to generate a pseudo-random number, which becomes the *rank* of the node. All nodes then propagate their ranks in the network. Finally, a node A views the m nodes with the m smallest ranks, among all the ranks that A has seen, as A's committee.

We draw a simple yet interesting connection between i) our setting of PoW without difficulty adjustment, and ii) their sleepy PoS setting. On the surface, these two settings are quite different. But if we view the PoW solution quality in our setting as the rank in their setting, then our PoW committee becomes the same as their rank-based committee. Because of this, they actually also face the same challenge of not being able to directly guarantee **[P3]** and **[P4]**, in their majority quorums. They then overcome this challenge, via a careful design of their protocol. Given this interesting connection, adapting their protocol/techniques directly enables us to overcome the issue of our 2D-quorums not guaranteeing **[P3]** and **[P4]**.

Hence, our adaptation is conceptually simple, overall. Nevertheless, our adaptation does involve some (tedious) details. For completeness, Appendix A and Appendix B will provide the full pseudo-code of our `Generalized_GC_01`() and `Generalized_GC_012`() subroutines. To be fully rigorous, there we will also provide complete and self-contained proofs, for the end guarantees of these two subroutines.

D. Design of `CommonCoin()`

We now explain the design of `CommonCoin`(). `CommonCoin`() should return a common coin to the nodes, which is allowed to be both biased and inconsistent. The only property we need is that `CommonCoin`() returns 0 on all nodes with some non-zero constant probability, and returns 1 on all nodes with some non-zero constant probability. The exact values of these probabilities only affect performance and not security. Usually, `CommonCoin`() is designed by first selecting a node as the *coin producer*. This coin producer will send to other nodes a locally-generated random bit as the value of the common coin. This coin producer may or may not be honest.

In our context, using the beacon as the randomness, with half probability, we choose this coin producer from PoS nodes, and with the remaining probability, from PoW nodes. Note that our design here is independent of the security region Υ that we intend to tolerate. To see why this works, we first prove a simple lemma:

Lemma 4: If security region Υ is not self-contradicting, and if $(f_w, f_s) \in \Upsilon$, then we must have either $f_w \leq \frac{1}{2}$ or $f_s \leq \frac{1}{2}$.

Proof: Prove by contradiction and assume that $f_w > \frac{1}{2}$ and $f_s > \frac{1}{2}$. Then we would have both $1 - f_w < f_w$ and $1 - f_s < f_s$. By definition of a region and since $(f_w, f_s) \in \Upsilon$, we then would have $(1 - f_w, 1 - f_s) \in \Upsilon$, contradicting with Υ being not self-contradicting. □

Intuitively, the lemma tells us that either half of the stakes are honest, or half of the computational power is honest. Hence, by giving half probability to each type of resource, intuitively we would have at least roughly $\frac{1}{4}$ probability of selecting an honest coin producer, as long as Υ is not self-contradicting.

Now to choose a PoS (or PoW) node as the coin producer, a node A computes `hash`(`beacon` | current round number | C_A's public key) for every node $C_A \in \mathbb{S}_A$ (or $C_A \in \mathbb{W}_A$). The C_A yielding the smallest hash value will be the coin producer from A's perspective. Note that C_A may be different for different A's. Finally, if A is a coin producer from its own perspective, then A will locally generate a uniformly random bit, and send that bit to all nodes in $\mathbb{W}_A \cup \mathbb{S}_A$.

We can now prove:

Lemma 5: If $|\mathbb{W}| \geq (1 - f_w)m$, $|\mathbb{S}| \geq (1 - f_s)m$, and if the security region Υ is not self-contradicting, then except for negligible probability with respect to m, we have:

- At least with constant positive probability, there exists a single honest node C such that `CommonCoin`() returns the bit b on all nodes in $\mathbb{W} \cup \mathbb{S}$, where b is the uniformly random bit generated by C.

Proof: See Appendix C. □

Comment on Lemma 5. Lemma 5 implies that with positive constant probability, `CommonCoin`() returns 0 on all nodes in $\mathbb{W} \cup \mathbb{S}$, and also with positive constant probability, `CommonCoin`() returns 1 on all nodes in $\mathbb{W} \cup \mathbb{S}$.

E. Formal Guarantees of `Generalized_BA_Core()`

We have explained the design of `Generalized_BA_Core()`, including the various building blocks. Now we formalize its end-to-end security guarantees:

Theorem 6: Consider any execution where the beacon chooses the block proposer from PoW nodes, all nodes in $\mathbb{W} \cup \mathbb{S}$ invoke `Generalized_BA_Core()`, $|\mathbb{W}| \geq (1 - f_w)m$, $|\mathbb{S}| \geq (1 - f_s)m$, and the security zone Υ is not self-contradicting. Then regardless of the behavior of other nodes, we must have:

- **Agreement.** Except for negligible probability with respect to m and k, all nodes in $\mathbb{W} \cup \mathbb{S}$ must have the same return value from `Generalized_BA_Core()`.
- **Validity.** Let L be the node with the best PoW solution among all honest and malicious nodes. If L is honest, then all nodes in $\mathbb{W} \cup \mathbb{S}$ must get x_L (i.e., the block that L wants to propose) as the return value from `Generalized_BA_Core()`.

Proof: See Appendix D. □

Comment on Theorem 6. Theorem 6 assumes that the block proposer is a PoW node. If the block proposer is a PoS node, a similar theorem can be proved using a similar proof.

VII. Implementation and Experimental Results

We have implemented a research prototype of our blockchain design. Our prototype enables us to quantify the performance of our design, via concrete experiments.

A. Implementation Details and Experimental Methodology

Our implementation is in Java. We use (loosely-synchronized) system time to determine the rounds, and use TCP for communication. We use 10 Linux machines in a local-area network as our test-bed. To scale up our experiments, we run 50 blockchain nodes on each machine, which gives total 500 nodes in our experiments. Due to constraints in CPU and physical memory, nodes on the same physical machine are implemented as separate Java threads. Of course, these nodes do not interact/communicate with each other through the shared memory. In our experiments, we let all 500 nodes be both stake holders (each holding 1 stake) and computational power holders (each holding 1 unit of computational power). Similar to some prior works [6], [9], the overlay network in our experiments is formed by letting each node connect to 4 (uniformly) random nodes, and let each node accept all incoming connections. On average, this gives each node 8 neighbors in the overlay topology. In certain steps in our protocol, a node needs to send/forward many digital signatures. We assume that these signatures can be aggregated via a suitable aggregate signature scheme (e.g., [26]).

Algorithm 2 in our design invokes `Generalized_GC_012()`, and `Generalized_GC_012()` directly operates on the blocks. This will cause the messages in `Generalized_GC_012()` to be rather large. Our implementation uses a simple optimization, where the invocation of `Generalized_GC_012()` is done over the hashes of the blocks, rather than over the blocks themselves. The actual blocks are propagated outside of `Generalized_GC_012()`. Appendix E gives the details of this simple optimization.

A round in our algorithm is composed of receiving messages and then sending messages. In some of our subroutines, the last round only involves the receiving of messages. Our implementation naturally combines such a round with the immediate next round in the algorithm. For block propagation, since sometimes a block can be large, we fragment a block into smaller pieces.

For simplicity, we have excluded the following aspects in our implementation which do not affect the experimental results: (1) Due to CPU constraints, we did not implement digital signatures, aggregate signatures, or secure hashes. Instead, we use dummy data with the same size to replace them. For the same reason, we did not implement the PoW solving phase. Instead, the PoW committee is formed, simply by choosing uniformly random PoW nodes. We apply a random ordering among the members in the PoW committee, as the ordering of PoW solution quality. (2) We did not implement transactions. Instead, we fill each block with garbage bits. Consequently, there is no transfer of stakes in our experiments. (3) We did not implement any beacon generation mechanism. Instead, we directly inject random beacons when needed.

TABLE II: Experimental parameters.

PoS committee size	500
PoW committee size	500
round duration	10 seconds
value of k in Algorithm 2	10
security region to tolerate	as in Figure 1(d)
block size (MB)	10, 25, 50, 75, 100
allocated time for block propagation (seconds)	69, 157, 292, 439, 576

Table II gives the various parameters used in our experiments. Our design allows us to choose the size of the blocks, and we will experiment with 10MB to 100MB. Our design allocates a fixed amount of time for block propagation. To determine how much time we should allocate, we run experiments to measure the time needed to propagate a block (of a given size) to all nodes, under our settings. We then pessimistically allocate 2 times of that amount, in our protocol (for that block size). For example, we observe that propagating a 10MB block takes 34.7 seconds, and hence we allocate 69 seconds. Note that even if the block fails to propagate to all nodes in 69 seconds, it will not affect safety/security. The only consequence is that the nodes may end up agreeing on an empty block.

B. Experimental Results

All our experimental results are averaged over 5 runs.

Our goal. The primary goal of our experiments is to determine whether our blockchain design can deliver reasonably good performance to end-users, in terms of *transaction throughput* and *block confirmation latency*. These two metrics are the key performance measures, from the end users' perspective. The most related prior work to our work is Fitzi et al. [16].

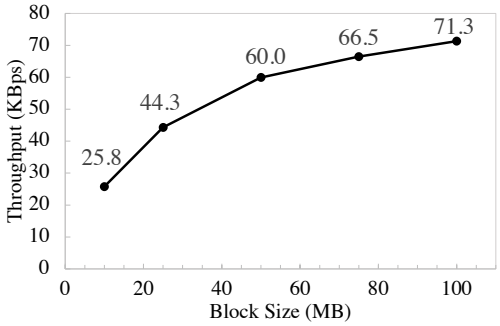


Fig. 3: End-to-end throughput.

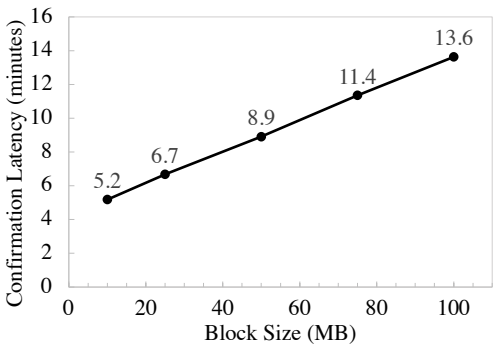


Fig. 4: Block confirmation latency.

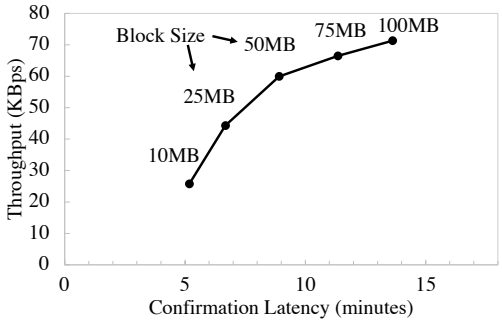


Fig. 5: Trade-off between throughput and latency.

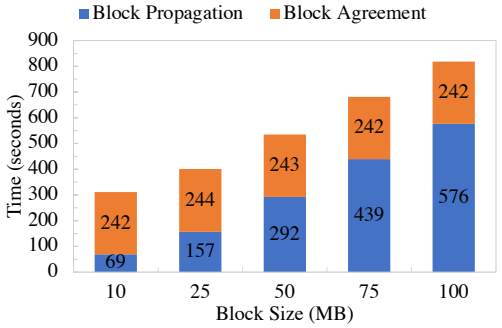


Fig. 6: Breakdown of confirmation latency.

Unfortunately, they did not give direct experimental results on throughput and confirmation latency. Also, their experiments involve only 4 nodes, rather than 500 nodes as in our experiments. This makes it impossible to do a direct comparison with [16]. Hence we will use Bitcoin [1] as a basic reference point, whose performance and usability are well-understood.

It is important to remember that the contribution of our work is *not* about better performance. Instead, our contribution is in achieving optimal resilience, and tolerating security regions that no existing protocol can tolerate. Hence, we only aim to show that our performance is reasonably good.

Throughput. Same as some prior works [6], [9], we define *transaction throughput* as the total size of all the blocks confirmed with a certain time period divided by the duration of that time period. Figure 3 plots the measured throughput of our blockchain, under different block sizes. As expected, the throughput of our blockchain increases with the block size. With appropriate block size (e.g., 50MB), our blockchain gives a good throughput of about 60KBps. Under a typical 0.5KB transaction size, this would translate to about 120 transactions/second. As a basic reference point, Bitcoin has a throughput of about 1.7KBps.

Confirmation latency. *Block confirmation latency* is the time needed, after a block is proposed, for the block to get confirmed (i.e., its position in the ledger to be finalized). Figure 4 plots the measured block confirmation latency, under different block sizes. The confirmation latency increases with the block size, since larger blocks take more time to propagate, and hence longer to confirm. The overall confirmation latency is reasonably good — for example, with 50MB block size, our confirmation latency is below 10 minutes. As a basic reference point, Bitcoin takes about 60 minutes to confirm a block, since Bitcoin generates one block every 10 minutes, and a block needs to be "6-blocks-deep" to get confirmed.

Deeper understanding. To gain some deeper understanding, Figure 5 combines Figure 3 and 4, to highlight the trade-off between throughput and confirmation latency. A larger block size enables our blockchain to deliver a higher throughout, but at the cost of larger block confirmation latency. Different applications hence may choose different trade-offs on this spectrum, by choosing the corresponding block sizes.

Figure 6 further provides the breakdown of the measured confirmation latency into two components. The first component is for block propagation, while the second one is for running the BA protocol and achieving agreement on the block. Th figure shows that the increase of confirmation latency under larger blocks is almost entirely due to the first component. This is as expected, since our BA protocol actually operates on the hash of the block (see Appendix E), and is insensitive to the block size.

VIII. Generalizing to Multiple Resources

Our design for combining PoW and PoS can be naturally generalized to combining more resources. The reason is that our blockchain design is largely independent of PoW and PoS — the only place where PoW and PoS are used is in the 2D-quorums. Now if we have $t > 2$ different resources, one could naturally generalize the 2D-quorums to t-dimensional quorums. The design of the BA protocol (other than the quorums) does not need to be changed at all.

IX. Conclusions

This paper has proposed the design, analysis, implementation, and evaluation of a novel blockchain protocol that combines PoS and PoW. Our blockchain offers *optimal resilience*, namely, we can tolerate every security region that is not self-contradicting. Our design also avoids PoW difficulty adjustment when combining PoW and PoS. To achieve these results, we use the key novel technique of 2D-quorums, which can also be of separate independent interest. Finally, our design naturally extends to combining more resources.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008, https://bitcoin.org/bitcoin.pdf.

[2] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "Bitcoin-NG: A Scalable Blockchain Protocol," in *NSDI*, 2016.

[3] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, 2014, https://ethereum.github.io/yellowpaper/paper.pdf.

[4] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *CCS*, 2016.

[5] V. Bagaria, S. Kannan, D. Tse, G. Fanti, and P. Viswanath, "Prism: Deconstructing the Blockchain to Approach Physical Limits," in *CCS*, 2019.

[6] H. Yu, I. Nikolic, R. Hou, and P. Saxena, "OHIE: Blockchain Scaling Made Simple," in *IEEE Symposium on Security and Privacy*, 2020.

[7] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *CRYPTO*, 2017.

[8] P. Daian, R. Pass, and E. Shi, "Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake," in *International Conference on Financial Cryptography and Data Security*, 2019.

[9] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *SOSP*, 2017.

[10] S. Park, A. Kwon, G. Fuchsbauer, P. Gaži, J. Alwen, and K. Pietrzak, "Spacemint: A cryptocurrency based on proofs of space," in *International Conference on Financial Cryptography and Data Security*, 2018.

[11] B. Cohen and K. Pietrzak, "The chia network blockchain," 2019, https://www.chia.net/assets/ChiaGreenPaper.pdf.

[12] T. Moran and I. Orlov, "Simple proofs of space-time and rational proofs of storage," in *Annual International Cryptology Conference*, 2019.

[13] S. King and S. Nadal, "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake," https://decred.org/research/king2012.pdf, 2012.

[14] I. Bentov, C. Lee, A. Mizrahi, and M. Rosenfeld, "Proof of activity: Extending bitcoin's proof of work via proof of stake [extended abstract]," *ACM SIGMETRICS Performance Evaluation Review*, vol. 42, no. 3, 2014.

[15] T. Duong, L. Fan, J. Katz, P. Thai, and H.-S. Zhou, "2-hop blockchain: Combining proof-of-work and proof-of-stake securely," in *ESORICS*, 2020.

[16] M. Fitzi, X. Wang, S. Kannan, A. Kiayias, N. Leonardos, P. Viswanath, and G. Wang, "Minotaur: Multi-resource blockchain consensus," in *CCS*, Nov. 2022.

[17] P. Thai, L. Njilla, T. Duong, L. Fan, and H.-S. Zhou, "A generic paradigm for blockchain design," in *International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, 2018.

[18] A. Chepurnoy, T. Duong, L. Fan, and H.-S. Zhou, "Twinscoin: A cryptocurrency via proof-of-work and proof-of-stake," in *ACM Workshop on Blockchains, Cryptocurrencies, and Contracts*, 2018.

[19] V. Buterin and V. Griffith, "Casper the friendly finality gadget," arXiv preprint arXiv:1710.09437, 2017.

[20] M. Zamani, M. Movahedi, and M. Raykova, "RapidChain: Scaling Blockchain via Full Sharding," in *CCS*, 2018.

[21] R. Hou, H. Yu, and P. Saxena, "Using throughput-centric byzantine broadcast to tolerate malicious majority in blockchains," in *IEEE Symposium on Security and Privacy*, 2022.

[22] V. Goyal, H. Li, and J. Raizes, "Instant block confirmation in the sleepy model," in *International Conference on Financial Cryptography and Data Security*, 2021.

[23] H. Li, "Generalized byzantine agreement with incomplete views," Master's thesis, Carnegie Mellon University Pittsburgh, PA, 2019.

[24] S. Micali and V. Vaikuntanathan, "Optimal and player-replaceable consensus with an honest majority," MIT CSAIL Technical Report, 2017-004, 2017.

[25] M. Mitzenmacher and E. Upfal, *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge university press, 2005.

[26] D. Boneh, M. Drijvers, and G. Neven, "Compact multi-signatures for smaller blockchains," in *ASIACRYPT*, 2018.

Appendix

A. Complete Pseudo-code and Proofs for `Generalized_GC_01()`

Algorithm 3 gives the complete pseudo-code for our `Generalized_GC_01()` subroutine, where the parts related to 2D-quorums have been highlighted. Throughout the pseudo-code, we view the id of a PoW node as containing both its public key and the best PoW solution that it has found. These PoW solutions are needed for Line 23.

High-level structure. We now explain the high-level internal structure of `Generalized_GC_01()`, which is the same as the $\{0,1\}$-Graded Broadcast protocol in [24].

In the first round, each node A sends its binary input value x_A, together with A's signature. Let us call such an A as a *broadcaster*. In the second round, each node A first receives all the incoming messages, which were sent in the first round. Next, A forwards each such message, after adding A's own signature (called a *counter-signature*).

The last round is the key. For each broadcaster B and a value x signed by B in the first round, node A checks two conditions: i) there are "sufficient" counter-signatures for B's signature on x (Line 20), and ii) there is no counter-signature for some other value $x' \neq x$ signed by B. If both conditions are met, then A views B as a *voter* for the value x. Finally, A will return a value x if there are "sufficient" voters for that x value (Line 24 and 25), or return the common coin if there is no such x value.

In the original design in [24], the condition of "sufficient" is simply defined to be "more than $\frac{n}{2}$". In our adaptation, we change this condition to be "having a 2D-quorum". The specifics of the 2D-quorum need to be different in different places, and we explain one by one.

2D-quorum at Line 20. At Line 20, for each broadcaster B and binary value x signed by B, node A checks whether there are "sufficient" counter-signatures for that. Let $G = \text{SigSet}_A(B, x)$ be the set of nodes doing these counter-signatures. We consider G as "sufficient" here, if G is a 2D-quorum for $\langle \mathbb{W}_A, \mathbb{S}_A \rangle$.

This will work because: i) by Theorem 3, the properties **[P1]** and **[P2]** must both hold for G; and ii) **[P1]** and **[P2]** are the only properties needed at Line 20. We will later formalize such arguments, by providing a complete proof.

2D-quorum at Line 24 and 25. At Line 24 (Line 25), node A checks whether the set of votes for the value 0 (1) is "sufficient". Here we need the 3 properties **[P1]**, **[P2]**, and **[P3]** to hold.

Algorithm 3 `Generalized_GC_01`(x_A). Let A denote the id of invoking node.

// first round
13: send $\langle x_A, \text{Sig}_A(x_A)\rangle$ to nodes in $\mathbb{W}_A \cup \mathbb{S}_A$;
// second round
14: receive messages from nodes in $\mathbb{W}_A \cup \mathbb{S}_A$, and ignore messages from other nodes;
15: **for** each $B \in \mathbb{W}_A \cup \mathbb{S}_A$: **if** $\langle x, \text{Sig}_B(x)\rangle$ was received for some x **then** send $\langle x, \text{Sig}_B(x), \text{Sig}_A(\text{Sig}_B(x))\rangle$ to nodes in $\mathbb{W}_A \cup \mathbb{S}_A$;
// third round
16: receive messages from nodes in $\mathbb{W}_A \cup \mathbb{S}_A$, and ignore messages from other nodes;
17: $\texttt{voters}_A[0] \leftarrow \emptyset$; $\texttt{voters}_A[1] \leftarrow \emptyset$;
18: **for** each B and each $x \in \{0, 1\}$ **do**
19: $\quad \text{SigSet}_A(B, x) \leftarrow \{C \mid$ message $\langle x, \text{Sig}_B(x), \text{Sig}_C(\text{Sig}_B(x))\rangle$ was received at Line 16$\}$;
20: $\quad$ **if** $\texttt{IsQuorum}(\text{SigSet}_A(B, x), \mathbb{W}_A, \mathbb{S}_A, \Upsilon)$ **and** (I did not receive $\langle x', \text{Sig}_B(x'), \text{Sig}_C(\text{Sig}_B(x'))\rangle$ for $\forall C \in \mathbb{W}_A \cup \mathbb{S}_A$ and $\forall x' \neq x$) **then** add B to $\texttt{voters}_A[x]$; // we say *this vote is due to* B
21: **endfor**
22: $\mathbb{V}_A \leftarrow \{B \mid B$ is a PoW node and Line 16 received message $\langle x, \text{Sig}_B(x), \text{Sig}_C(\text{Sig}_B(x))\rangle$ with any $C \in \mathbb{W}_A \cup \mathbb{S}_A$ and any $x\}$;
23: **if** $|\mathbb{V}_A| > m$ **then** keep the top m nodes in terms of PoW solution quality and remove all other nodes from $\mathbb{V}_A$;
24: **if** $\texttt{IsQuorum}(\texttt{voters}_A[0], \mathbb{V}_A, \mathbb{S}_A, \Upsilon)$ **then return** 0;
25: **elseif** $\texttt{IsQuorum}(\texttt{voters}_A[1], \mathbb{V}_A, \mathbb{S}_A, \Upsilon)$ **return** 1;
26: **else return** `CommonCoin`(); **endif**

Recall that **[P3]** means that if one honest node A believes that a set G of votes is "sufficient" for 0, and if another honest node D believe that a set G' is "sufficient" for 1, then G and G' must intersect. Such intersection prevents A from seeing a 2D-quorum for 0, when D sees a 2D-quorum for 1. Figure 2 earlier illustrated that simply requiring G (G') to be a 2D-quorum for $\langle \mathbb{W}_A, \mathbb{S}_A\rangle$ ($\langle \mathbb{W}_D, \mathbb{S}_D\rangle$) will not work — namely, G and G' might not intersect.

To deal with this, each node A constructs the set $\mathbb{V}_A$, which contains all those nodes B, where B is a PoW node and where A has seen at least one counter-signature for B's signature. If $|\mathbb{V}_A| > m$, we further truncate $\mathbb{V}_A$, by only keeping the top m nodes (in terms of PoW quality) in $\mathbb{V}_A$. Note that $\mathbb{V}_A$ may still be different for different A's. Line 24 now requires G (specifically, $\texttt{voters}_A[0]$) to be a 2D-quorum for $\langle \mathbb{V}_A, \mathbb{S}_A\rangle$. The requirement at Line 25 is similar.

End guarantees of `Generalized_GC_01`(). Theorem 7 next proves the agreement property of `Generalized_GC_01`():

Theorem 7: Consider any execution where all nodes in $\mathbb{W} \cup \mathbb{S}$ invoke `Generalized_GC_01`(), $|\mathbb{W}| \geq (1 - f_w)m$, $|\mathbb{S}| \geq (1 - f_s)m$, and the region Υ is not self-contradicting. Then regardless of the behavior of other nodes, there can never be two nodes $A_1 \in \mathbb{W} \cup \mathbb{S}$ and $A_2 \in \mathbb{W} \cup \mathbb{S}$ such that A_1 returns at Line 24 while A_2 returns at Line 25.

Proof: Prove by contradiction, and assume that such A_1 and A_2 exist. Consider the value of $\mathbb{V}_{A_1}$ and $\mathbb{V}_{A_2}$ at Line 24, and define $\mathbb{V}_{A_1A_2} = \mathbb{V}_{A_1} \cap \mathbb{V}_{A_2}$. Imagine that the nodes in $\mathbb{V}_{A_1}$ are sorted: The primary sorting key is whether the node belongs to $\mathbb{V}_{A_1A_2}$, and nodes in $\mathbb{V}_{A_1A_2}$ are ordered before nodes not in $\mathbb{V}_{A_1A_2}$. The secondary sorting key is the quality of the node's PoW solution, in decreasing order. We use $\mathbb{V}_{A_1}[i]$ to note the ith element in $\mathbb{V}_{A_1}$ after sorting. Similarly define $\mathbb{V}_{A_2}[i]$.

Consider any $i \in [1, m]$, and let $B_1 = \mathbb{V}_{A_1}[i]$ and $B_2 = \mathbb{V}_{A_2}[i]$. We claim that it is impossible for A_1 to add B_1 to $\texttt{voters}_{A_1}[0]$, if A_2 adds B_2 to $\texttt{voters}_{A_2}[1]$. We prove this claim via a simple contradiction. Since A_1 adds B_1 to $\texttt{voters}_{A_1}[0]$, the condition $\texttt{IsQuorum}(\text{SigSet}_{A_1}(B_1, 0), \mathbb{W}_{A_1}, \mathbb{S}_{A_1}, \Upsilon)$ must hold. By **[P2]** in Theorem 3, there must exist some node $C_1 \in \mathbb{W} \cup \mathbb{S}$, such that $C_1 \in \text{SigSet}_{A_1}(B_1, 0)$ and C_1 sends A_1 the message $\langle 0, \text{Sig}_{B_1}(0), \text{Sig}_{C_1}(\text{Sig}_{B_1}(0))\rangle$ at Line 15. Since $C_1 \in \mathbb{W} \cup \mathbb{S}$ and is honest, C_1 must also send this message to A_2 at Line 15. Similarly, there must also exist $C_2 \in \mathbb{W} \cup \mathbb{S}$ such that C_2 sends A_1 the message $\langle 1, \text{Sig}_{B_2}(1), \text{Sig}_{C_2}(\text{Sig}_{B_2}(1)\rangle$ at Line 15. We consider two cases:

- $B_1 \in \mathbb{V}_{A_1A_2}$. In this case, we must have $B_1 = B_2$. After A_1 receives the message from C_2 at Line 16, the second part of the condition at Line 20 makes it impossible for A_1 to adds B_1 to $\texttt{voters}_{A_1}[0]$.
- $B_1 \notin \mathbb{V}_{A_1A_2}$. In this case, we must have $B_2 \notin \mathbb{V}_{A_1A_2}$ and $B_1 \neq B_2$. At Line 22, A_1 must add B_2 to $\mathbb{V}_{A_1}$, due to the message from C_2. Similarly, A_2 must add B_1 to $\mathbb{V}_{A_2}$ at Line 22. If B_1's PoW solution is better than B_2's, then since $B_2 \in \mathbb{V}_{A_2}$, node B_1 should also be in $\mathbb{V}_{A_2}$. This contradicts with $B_1 \notin \mathbb{V}_{A_1A_2}$. A similar contradiction arises if B_2's PoW solution is better than B_1's.

We have proved our claim that $\forall i \in [1, m]$, it is impossible for A_1 to add $\mathbb{V}_{A_1}[i]$ to $\texttt{voters}_{A_1}[0]$, if A_2 adds $\mathbb{V}_{A_2}[i]$ to $\texttt{voters}_{A_2}[1]$. A similar reasoning also shows that it is impossible for A_1 to add $\mathbb{S}_{A_1}[i]$ to $\texttt{voters}_{A_1}[0]$, if A_2 adds $\mathbb{S}_{A_2}[i]$ to $\texttt{voters}_{A_2}[1]$.

Next, define $\texttt{countW}_0 = |\texttt{voters}_{A_1}[0] \cap \mathbb{V}_{A_1}|$, $\texttt{countS}_0 = |\texttt{voters}_{A_1}[0] \cap \mathbb{S}_{A_1}|$, $\texttt{countW}_1 = |\texttt{voters}_{A_2}[1] \cap \mathbb{V}_{A_2}|$, and $\texttt{countS}_1 = |\texttt{voters}_{A_2}[1] \cap \mathbb{S}_{A_2}|$. Our earlier claim then immediately implies that $\texttt{countW}_0 + \texttt{countW}_1 \leq m$ and $\texttt{countS}_0 + \texttt{countS}_1 \leq m$.

Since A_1 returns at Line 24, the condition $\texttt{IsQuorum}(\texttt{voters}_{A_1}[0], \mathbb{U}_{A_1}, \mathbb{S}_{A_1}, \Upsilon)$ must hold. This means:

$$(1 - \frac{\texttt{countW}_0}{m}, 1 - \frac{\texttt{countS}_0}{m}) \in \Upsilon \qquad (2)$$

Since A_2 returns at Line 25, similar as above, we have $(1 - \frac{\texttt{countW}_1}{m}, 1 - \frac{\texttt{countS}_1}{m}) \in \Upsilon$. Because $\texttt{countW}_0 + \texttt{countW}_1 \leq m$

and $\texttt{countS}_0 + \texttt{countS}_1 \leq m$, we know that $\frac{\texttt{countW}_0}{m} \leq 1 - \frac{\texttt{countW}_1}{m}$ and $\frac{\texttt{countS}_0}{m} \leq 1 - \frac{\texttt{countS}_1}{m}$. By definition of a region, since $(1 - \frac{\texttt{countW}_1}{m}, 1 - \frac{\texttt{countS}_1}{m}) \in \Upsilon$, we then must also have:

$$(\frac{\texttt{countW}_0}{m}, \frac{\texttt{countS}_0}{m}) \quad \in \quad \Upsilon \tag{3}$$

But with Equation 2 and 3, the region Υ becomes self-conflicting. □

Theorem 8 next proves the validity property of Generalized_GC_01():

Theorem 8: Consider any execution where all nodes in $\mathbb{W} \cup \mathbb{S}$ invoke Generalized_GC_01(b) with the same b value, $|\mathbb{W}| \geq (1 - f_w)m$, $|\mathbb{S}| \geq (1 - f_s)m$, and the region Υ is not self-contradicting. Then regardless of the behavior of other nodes, all nodes in $\mathbb{W} \cup \mathbb{S}$ must return that b value.

Proof: Consider any two nodes A and B in $(\mathbb{W} \cup \mathbb{S})$. At Line 20, $\text{SigSet}_A(B, b)$ is guaranteed to contain all nodes in $\mathbb{W} \cup \mathbb{S}$. By property **[P1]** in Theorem 3, $\texttt{IsQuorum}(\text{SigSet}_A(B, b), \mathbb{W}_A, \mathbb{S}_A, \Upsilon)$ must hold. Together with the fact that B never signed any other value $b' \neq b$, we know that the condition at Line 20 must be satisfied. Then A must add B to $\texttt{voters}_A[b]$. Furthermore, we must also have $B \in \mathbb{V}_A \cup \mathbb{S}_A$ after Line 23.

The above holds for all $B \in \mathbb{W} \cup \mathbb{S}$, which means $(\mathbb{W} \cup \mathbb{S}) \subseteq \texttt{voters}_A[b]$ and $(\mathbb{W} \cup \mathbb{S}) \subseteq (\mathbb{V}_A \cup \mathbb{S}_A)$. We next prove that:

- This makes $\texttt{voters}_A[b]$ a 2D-quorum for $\langle \mathbb{V}_A, \mathbb{S}_A \rangle$ (which is essentially the property **[P1]**): We have $1 - \frac{|\texttt{voters}_A[b] \cap \mathbb{V}_A|}{m} \leq 1 - \frac{|\mathbb{W}|}{m} \leq f_w$ and $1 - \frac{|\texttt{voters}_A[b] \cap \mathbb{S}_A|}{m} \leq f_s$. Hence given $(f_w, f_s) \in \Upsilon$, we must also have $(1 - \frac{|\texttt{voters}_A[b] \cap \mathbb{W}_A|}{m}, 1 - \frac{|\texttt{voters}_A[b] \cap \mathbb{S}_A|}{m}) \in \Upsilon$. Thus $\texttt{voters}_A[b]$ is a 2D-quorum for $\langle \mathbb{V}_A, \mathbb{S}_A \rangle$.
- This makes it impossible for $\texttt{voters}_A[b']$ to be a 2D-quorum for $\langle \mathbb{V}_A, \mathbb{S}_A \rangle$ for any $b' \neq b$ (which is essentially property **[P2]**): For all $B \in \mathbb{W} \cup \mathbb{S}$, node A will never add B to $\texttt{voters}_A[b']$. Hence $(\mathbb{W} \cup \mathbb{S}) \cap \texttt{voters}_A[b'] = \emptyset$. Assume, for the purpose of contradiction, that $\texttt{voters}_A[b']$ is indeed a 2D-quorum for $\langle \mathbb{V}_A, \mathbb{S}_A \rangle$. This implies $(1 - \frac{|\texttt{voters}_A[b'] \cap \mathbb{V}_A|}{m}, 1 - \frac{|\texttt{voters}_A[b'] \cap \mathbb{S}_A|}{m}) \in \Upsilon$. Since $|\mathbb{W}| \geq (1 - f_w)m$ and $\mathbb{W} \subseteq \mathbb{V}_A$, we know that $1 - f_w = 1 - \frac{m - (1 - f_w)m}{m} \leq 1 - \frac{|\texttt{voters}_A[b'] \cap \mathbb{V}_A|}{m}$. Similarly, $1 - f_s \leq 1 - \frac{|\texttt{voters}_A[b'] \cap \mathbb{S}_A|}{m}$. By definition of a region, we then must have $(1 - f_w, 1 - f_s) \in \Upsilon$. Together with the fact that $(f_w, f_s) \in \Upsilon$, the region Υ now becomes self-contradicting.

From the above two claims, as well as the conditions at Line 24 and 25, we know that every node $A \in \mathbb{W} \cup \mathbb{S}$ must return b. □

B. Complete Pseudo-code and Proofs for Generalized_GC_012*()*

Choosing a block proposer. In Generalized_BA_Core(), before Generalized_GC_012() is invoked, each node A needs to use the beacon to choose a *block proposer* for the next block on the blockchain. We first discuss how this is done. To choose a block proposer, A first determines whether the block proposer should be a PoW node or a PoS node, by using the beacon as randomness. The proposer should be a PoS node with half probability, and a PoW node with the remaining half probability. If the proposer should be a PoS node, then A further uses the beacon as randomness to choose a uniformly random node from $\mathbb{S}_A$, as A's view of the block proposer (denoted as L_A). If the proposer should be a PoW node, then A will pick the node in $\mathbb{W}_A$ that has the best-quality PoW solution as L_A. Note that L_A may or may not be an honest node, and L_A may be different for different A's.

In the next, we will discuss Generalized_GC_012() while assuming that the block proposer is a PoW node. The case when L_A is a PoS node is similar (in fact simpler).

Pseudo-code for Generalized_GC_012(). Algorithm 4 gives the complete pseudo-code for our Generalized_GC_012() subroutine, where the parts related to 2D-quorums have been highlighted. Again, we view the id of a PoW node as containing both its public key and the best PoW solution that it has found.

Generalized_GC_012() is adapted from the $\{0, 1, 2\}$-Graded Broadcast protocol in [24]. While $\{0, 1, 2\}$-Graded Broadcast is more complex than $\{0, 1\}$-Graded Broadcast, our adaptation is largely unrelated to other mechanisms in the protocol. Hence we will focus on our adaptation, namely, how we use 2D-quorums. We refer interested readers to [24] for the overall structure and logic of Generalized_GC_012(), which are directly adopted from [24]. To be fully rigorous and for completeness, we will later provide a self-contained proof for Generalized_GC_012().

2D-quorum at Line 40 and 49. Here we require $\text{SigSet}_A(L_A, x)$ or $\texttt{voters}_A(L_A, x)$ to be a 2D-quorum for $\langle \mathbb{W}_A, \mathbb{S}_A \rangle$. This is the same idea as Line 20 earlier, and we do not further discuss.

2D-quorum at Line 48, 50, 52, and 53. These lines are all concerned with determining whether a set G (e.g., $G = \text{SigSetFull}_Z(L_A, x)$) is "sufficient". By the original design in [24], here we need property **[P4]** to hold: Namely, if one honest node A believes that G is "sufficient" from A's perspective, then we need every other honest node B to also believe that G is "sufficient" from B's perspective. How to guarantee **[P4]** is not immediately obvious, since A and B can have different committees.

But this requirement is only asymmetric. Specifically, the original design in [24] only needs the following: If A believes that G is "sufficient" at Line 48, then B needs to believe that G is "sufficient" at Line 50, 52, and 53. Hence, a stronger definition can be used at Line 48, and a weaker definition can be used at Line 50, 52, and 53.

Specifically, we first define a simple procedure Merge(), which takes two sets (X_1 and X_2) of PoW nodes as input. The procedure sorts all the nodes in $(X_1 \cup X_2)$ in decreasing order of PoW solution quality. The procedure then returns the first m nodes in that ordering, or all nodes if $|X_1 \cup X_2| < m$.

Algorithm 4 `Generalized_GC_012`(L_A, x_A). Let A denote the id of invoking node.

// first round
27: let L_A be the node with the best PoW solution in $\mathbb{W}_A$; let x_A be the block that A wants to propose;
28: $\texttt{flag}_A \leftarrow \texttt{false}$;
29: send $\langle \text{Proposer\_Id}, L_A \rangle$ to nodes in $\mathbb{W}_A \cup \mathbb{S}_A$;
// second round
30: receive messages from nodes in $\mathbb{W}_A \cup \mathbb{S}_A$, and ignore messages from other nodes;
31: $\texttt{flag}_A \leftarrow \texttt{true}$, **if** any message $\langle \text{Proposer\_Id}, B \rangle$ is received with B's PoW solution being better than L_A's;
32: **if** ($\texttt{flag}_A = \texttt{false}$) and ($A = L_A$) **then** send $\langle x_A, \text{Sig}_A(x_A) \rangle$ to nodes in $\mathbb{W}_A \cup \mathbb{S}_A$; **endif**
// third round
33: receive messages from nodes in $\mathbb{W}_A \cup \mathbb{S}_A$, and ignore messages from other nodes;
34: **if** ($\texttt{flag}_A = \texttt{false}$) and (message $\langle x, \text{Sig}_{L_A}(x) \rangle$ was received for any x) **then**
35: send $\langle x, \text{Sig}_{L_A}(x), \text{Sig}_A(\text{Sig}_{L_A}(x)) \rangle$ to all nodes in $\mathbb{W}_A \cup \mathbb{S}_A$; // pick any one if there are multiple such x
36: **endif**
// fourth round
37: receive messages from nodes in $\mathbb{W}_A \cup \mathbb{S}_A$, and ignore messages from other nodes;
38: $\text{SigSetFull}_A(L_A, x) \leftarrow \{\text{Sig}_Z(\text{Sig}_{L_A}(x)) \mid$ message $\langle x, \text{Sig}_{L_A}(x), \text{Sig}_Z(\text{Sig}_{L_A}(x)) \rangle$was received$\}$;
39: // In all the following, $\text{SigSet}_A(L_A, x)$ is the set of signers (i.e., the Z's) in $\text{SigSetFull}_A(L_A, x)$.
40: **if** ($\texttt{flag}_A = \texttt{false}$) **and** ($\texttt{IsQuorum}(\text{SigSet}_A(L_A, x), \mathbb{W}_A, \mathbb{S}_A, \Upsilon)$ holds for some x) // pick any one if there are multiple such x
and (Line 37 did not receive any $\langle x', \text{Sig}_{L_A}(x'), \text{Sig}_{Z'}(\text{Sig}_{L_A}(x')) \rangle$ for $\forall Z'$ and $\forall x' \neq x$) **then**
41: send $\langle x, \text{SigSetFull}_A(L_A, x) \rangle$ to nodes in $\mathbb{W}_A \cup \mathbb{S}_A$;
42: **endif**
43: send $\langle \text{To\_Merge}, \mathbb{W}_A \rangle$ to nodes in $\mathbb{W}_A \cup \mathbb{S}_A$;
// fifth round
44: receive messages from nodes in $\mathbb{W}_A \cup \mathbb{S}_A$, and discard messages from other nodes;
45: $\mathbb{U}_A \leftarrow \mathbb{W}_A$; **for** each $\langle \text{To\_Merge}, \mathbb{W}_B \rangle$ message received at Line 44 **do** $\mathbb{U}_A \leftarrow \texttt{Merge}(\mathbb{U}_A, \mathbb{W}_B)$;
46: We say that a set G of nodes is a *strong quorum for* A if $\texttt{IsQuorum}(G, \texttt{Merge}(\mathbb{U}_A, G), \mathbb{S}_A, \Upsilon)$ holds;
47: We say that a set G of nodes is a *weak quorum for* A if $\texttt{IsQuorum}(G, \texttt{Merge}(\mathbb{W}_A, G), \mathbb{S}_A, \Upsilon)$ holds;
48: $\texttt{voters}_A(L_A, x) \leftarrow \{Z \mid \langle x, \text{SigSetFull}_Z(L_A, x) \rangle$ was received at Line 44 with $\text{SigSet}_Z(L_A, x)$ being a strong quorum for $A\}$
49: **if** (for some x, we have $\texttt{IsQuorum}(\texttt{voters}_A(L_A, x), \mathbb{W}_A, \mathbb{S}_A, \Upsilon)$) **and**
50: (Line 44 did not receive $\langle x', \text{SigSetFull}_{Z'}(L', x') \rangle$ for $\forall L', \forall Z', \forall x' \neq x$, with $\text{SigSet}_{Z'}(L', x')$ being a weak quorum for A) **then**
51: **return** $\langle 2, x \rangle$; // use any x value here if there are multiple such x
52: **elseif** (Line 44 received $\langle x, \text{SigSetFull}_Z(L, x) \rangle$ for some L and Z and x, with $\text{SigSet}_Z(L, x)$ being a weak quorum for A) **and**
53: (Line 44 did not receive $\langle x', \text{SigSetFull}_{Z'}(L', x') \rangle$ for $\forall L', \forall Z', \forall x' \neq x$, with $\text{SigSet}_{Z'}(L', x')$ being a weak quorum for A) **then**
54: **return** $\langle 1, x \rangle$; // use any x value here if there are multiple such x
55: **else**
56: **return** $\langle 0, 0 \rangle$;
57: **endif**

Now for the weaker requirement at Line 50, 52, and 53 on node B, we require G to be a 2D-quorum for $\langle \texttt{Merge}(\mathbb{W}_B, G), \mathbb{S}_B \rangle$. We also call such a 2D-quorum as a *weak quorum*.

To prepare for the stronger requirement, we let every node C send its $\mathbb{W}_C$ to other nodes. Now node A will use `Merge()` to merge all the $\mathbb{W}_C$'s received, as well as its own $\mathbb{W}_A$. Let the result of the merge be $\mathbb{U}_A$. (Note that $\mathbb{U}_A$ can be different for different A's.) For the stronger requirement at Line 48 on node A, we require G to be a 2D-quorum for $\langle \texttt{Merge}(\mathbb{U}_A, G), \mathbb{S}_A \rangle$. We also call such a 2D-quorum as a *strong quorum*.

End guarantees of `Generalized_GC_012()`. Before proving the end guarantees of `Generalized_GC_012()`, we first prove two lemmas.

Lemma 9: **(Variant of [P4] for strong/weak quorum.)** If $|\mathbb{W}| \geq (1 - f_w)m$, $|\mathbb{S}| \geq (1 - f_s)m$, and the region Υ is not self-contradicting, then in Algorithm 4, a strong quorum G for an honest node $A \in (\mathbb{W} \cup \mathbb{S})$ at Line 48 must be a weak quorum for every honest node $B \in (\mathbb{W} \cup \mathbb{S})$ at Line 50, 52, and 53.

Proof: We know that G is a 2D-quorum for $\langle \texttt{Merge}(\mathbb{U}_A, G), \mathbb{S}_A \rangle$, and we need to prove that G is a 2D-quorum for $\langle \texttt{Merge}(\mathbb{W}_B, G), \mathbb{S}_B \rangle$. It suffices to prove that $G_1 \subseteq G_2$, where $G_1 = G \cap \texttt{Merge}(\mathbb{U}_A, G)$ and $G_2 = G \cap \texttt{Merge}(\mathbb{W}_B, G)$. Let X be any node in G_1. This implies $X \in \texttt{Merge}(\mathbb{U}_A, G)$, and that the number of nodes in $(\mathbb{U}_A \cup G)$ with better PoW solutions than X is at most $m-1$. Hence the number of nodes in $(\mathbb{W}_B \cup G)$ with better PoW solutions than X is also at most $m - 1$. Together with the fact that $X \in G \subseteq (\mathbb{W}_B \cup G)$, we have $X \in \texttt{Merge}(\mathbb{W}_B, G)$ and in turn $X \in G_2$. □

Lemma 10: **([P1] and [P2] for strong/weak quorum.)** If $|\mathbb{W}| \geq (1 - f_w)m$, $|\mathbb{S}| \geq (1 - f_s)m$, and the region Υ is not self-contradicting, then properties **[P1]** and **[P2]** must hold:

- At Line 48, where $\mathbb{T}_w^A = \texttt{Merge}(\mathbb{U}_A, G)$ and $\mathbb{T}_s^A = \mathbb{S}_A$.
- At Line 50, 52, and 53, where $\mathbb{T}_w^A = \texttt{Merge}(\mathbb{W}_A, G)$ and $\mathbb{T}_s^A = \mathbb{S}_A$.

Proof: We first prove **[P1]** at Line 48. Let $G = \mathbb{W} \cup \mathbb{S}$. We must have $\mathbb{W} \subseteq \texttt{Merge}(\mathbb{U}_A, G)$. Then we have $\mathbb{W} \subseteq (G \cap$

$\texttt{Merge}(\mathbb{U}_A, G)) = G \cap \mathbb{T}_w^A$. In turn, we have $1 - \frac{|G \cap \mathbb{T}_w^A|}{m} \leq 1 - \frac{|\mathbb{W}|}{m} \leq f_w$. Similarly, $1 - \frac{|G \cap \mathbb{S}_A|}{m} \leq f_s$. Hence given $(f_w, f_s) \in \Upsilon$, we must also have $(1 - \frac{|G \cap \mathbb{T}_w^A|}{m}, 1 - \frac{|G \cap \mathbb{T}_s^A|}{m}) \in \Upsilon$. Thus property **[P1]** holds at Line 48. A similar proof will also show that **[P1]** holds at Line 50, 52, and 53.

We next prove **[P2]** at Line 48 via a contradiction, and assume that $G \cap (\mathbb{W} \cup \mathbb{S}) = \emptyset$ and that G is a 2D-quorum for $\langle \texttt{Merge}(\mathbb{U}_A, G), \mathbb{S}_A \rangle$. This implies $(1 - \frac{|G \cap \texttt{Merge}(\mathbb{U}_A, G)|}{m}, 1 - \frac{|G \cap \mathbb{S}_A|}{m}) \in \Upsilon$. Since $|\mathbb{W}| \geq (1 - f_w)m$ and $\mathbb{W} \subseteq \texttt{Merge}(\mathbb{U}_A, G)$, we know that $1 - f_w = 1 - \frac{m-(1-f_w)m}{m} \leq 1 - \frac{|G \cap \texttt{Merge}(\mathbb{U}_A, G)|}{m}$. Similarly, $1 - f_s \leq 1 - \frac{|G \cap \mathbb{S}_A|}{m}$. By definition of a region, we then must have $(1 - f_w, 1 - f_s) \in \Upsilon$. Together with the fact that $(f_w, f_s) \in \Upsilon$, the region Υ now becomes self-contradicting. A similar proof will also show that **[P2]** holds at Line 50, 52, and 53. □

Theorem 11 next proves the agreement property of `Generalized_GC_012()`:

Theorem 11: Consider any execution where all nodes in $\mathbb{W} \cup \mathbb{S}$ invoke `Generalized_GC_012()`, $|\mathbb{W}| \geq (1 - f_w)m$, $|\mathbb{S}| \geq (1 - f_s)m$, and the zone Υ is not self-contradicting. Then regardless of the behavior of other nodes, if some node $A \in \mathbb{W} \cup \mathbb{S}$ returns $\langle 2, v \rangle$ for some v, then each node $B \in \mathbb{W} \cup \mathbb{S}$ must return either $\langle 2, v \rangle$ or $\langle 1, v \rangle$.

Proof: Since A returns $\langle 2, v \rangle$, the condition $\texttt{IsQuorum}(\texttt{voters}_A(L_A, x), \mathbb{W}_A, \mathbb{S}_A, \Upsilon)$ must hold at Line 49. By property **[P2]** in Theorem 3, the set $\texttt{voters}_A(L_A, v)$ must contain some (honest) node $C \in \mathbb{W} \cup \mathbb{S}$, such that:

- C sent $\langle v, \text{SigSetFull}_C(L_A, v) \rangle$ to all nodes in $\mathbb{W}_C \cup \mathbb{S}_C$ at Line 41, and
- $\text{SigSet}_C(L_A, v)$ is a strong quorum for A at Line 48.

Because $B \in (\mathbb{W} \cup \mathbb{S}) \subseteq (\mathbb{W}_C \cup \mathbb{S}_C)$, node B must also have received this message $\langle v, \text{SigSetFull}_C(L_A, v) \rangle$ at Line 44. By Lemma 9, $\text{SigSet}_C(L_A, v)$ must be a weak quorum for B at Line 50, 52, and 53.

We further claim that at Line 53, B must have not seen any $\text{SigSet}_{Z'}(L', x')$ for $\forall L'$, $\forall Z'$, $\forall x' \neq v$, such that $\text{SigSet}_{Z'}(L', x')$ is a weak quorum for B. If this claim indeed holds, then we are done because:

- If B does return at Line 51, then since B has received $\langle v, \text{SigSetFull}_C(L_A, v) \rangle$ and since $\text{SigSet}_C(L_A, v)$ is a weak quorum for B at Line 50, the only possible return value is $\langle 2, v \rangle$.
- If B does not return at Line 51, then both the conditions at Line 52 and 53 are guaranteed to be satisfied, and B will return $\langle 1, v \rangle$.

We now prove the earlier claim, via a contradiction. Assume that at Line 53, node B has seen $\text{SigSetFull}_{Z'}(L', x')$ for some L', Z', and $x' \neq v$, such that $\text{SigSet}_{Z'}(L', x')$ is a weak quorum for B. By property **[P2]** in Lemma 10, the set $\text{SigSet}_{Z'}(L', x')$ must contain $\text{Sig}_D(\text{Sig}_{L'}(x'))$ for some (honest) node $D \in \mathbb{W} \cup \mathbb{S}$. This means that node D must have sent $\langle x', \text{Sig}_{L'}(x'), \text{Sig}_D(\text{Sig}_{L'}(x')) \rangle$ to all nodes in $\mathbb{W}_D \cup \mathbb{S}_D$ at Line 35. This also implies $L_D = L'$.

Since $C \in (\mathbb{W} \cup \mathbb{S}) \subseteq (\mathbb{W}_D \cup \mathbb{S}_D)$, node C must also receive this message at Line 37. Furthermore, given that C sends a message at Line 41 and D sends a message at Line 35, we must have $\texttt{flag}_C = \texttt{flag}_D = \texttt{false}$. We hence must have $L_C = L_D$, since otherwise at least one of these two flags would have been set to be `true`. Then since $L_D = L'$, we have $L_C = L'$. This means that at Line 40, node C has seen $\langle x', \text{Sig}_{L_C}(x'), \text{Sig}_D(\text{Sig}_{L_C}(x')) \rangle$ with $x' \neq v$. This would then prevent C from sending out $\langle v, \text{SigSetFull}_C(L_A, v) \rangle$ at Line 41, which is a contradiction. □

Theorem 12 next proves the validity property of `Generalized_GC_012()`:

Theorem 12: Let L be the node with the best PoW solution among all honest and malicious nodes. If L is an honest node, $|\mathbb{W}| \geq (1 - f_w)m$, $|\mathbb{S}| \geq (1 - f_s)m$, zone Υ is not self-contradicting, and every node $A \in \mathbb{W} \cup \mathbb{S}$ invokes `Generalized_GC_012`(L_A, x_A) with $L_A = L$ and some x_A, then every honest node in $\mathbb{W} \cup \mathbb{S}$ must return $(2, x_L)$.

Proof: It is easy to see that for all node $A \in \mathbb{W} \cup \mathbb{S}$, the variable $\texttt{flag}_A$ will never be set to be `true`. Next, all node $A \in \mathbb{W} \cup \mathbb{S}$ must receive the message $\langle x_L, \text{Sig}_L(x_L) \rangle$ at Line 33. Since $L_A = L$, node A must further send the message $\langle x_L, \text{Sig}_L(x_L), \text{Sig}_A(\text{Sig}_L(x_L)) \rangle$ at Line 35.

This in turn means that later at Line 38, every node in $\mathbb{W} \cup \mathbb{S}$ will be included in the set $\text{SigSet}_A(L, x_L)$. In turn, by property **[P1]** in Theorem 3, $\texttt{IsQuorum}(\text{SigSet}_A(L, x_L), \mathbb{W}_A, \mathbb{S}_A, \Upsilon)$ must hold on node A at Line 40. Because node L has never signed on any value other than x_L, node A must satisfy all the conditions at Line 40 and will send the message $\langle x_L, \text{SigSetFull}_A(L, x_L) \rangle$.

Furthermore, at Line 48, for any given $Z \in \mathbb{W} \cup \mathbb{S}$, the set $\text{SigSet}_Z(L, x_L)$ must be superset of $\mathbb{W} \cup \mathbb{S}$. By property **[P1]** in Lemma 10, $\text{SigSet}_Z(L, x_L)$ must be a strong quorum for A, and hence $Z \in \texttt{voters}_A(L, x_L)$ at Line 48. Now since $(\mathbb{W} \cup \mathbb{S}) \subseteq \texttt{voters}_A(L, x_L)$, by property **[P1]** in Theorem 3, we know that $\texttt{voters}_A(L, x_L)$ must be a 2D-quorum for $\langle \mathbb{W}_A, \mathbb{S}_A \rangle$ at Line 49.

Finally, each honest node $Z \in \mathbb{W} \cup \mathbb{S}$ sends at most a single message at Line 35, which is $\langle x_L, \text{Sig}_L(x_L), \text{Sig}_Z(\text{Sig}_L(x_L)) \rangle$. Even if every node $M \notin \mathbb{W} \cup \mathbb{S}$ signs and sends a message $\langle x', \text{Sig}_{L'}(x'), \text{Sig}_M(\text{Sig}_{L'}(x')) \rangle$ for some L' and some $x' \neq x_L$ at Line 35, by property **[P2]** in Lemma 10, those messages will still not be sufficient to make $\text{SigSet}_{Z'}(L', x')$ a weak quorum for A at Line 50.

Hence node A must satisfy both of the conditions at Line 49 and 50, and must return $(2, x_L)$. □

Comment on Theorem 12. Theorem 12 is for the case where the block proposer is a PoW node. A similar theorem can be proved for the case where the block proposer is a PoS node.

C. Proof for Lemma 5

Proof for Lemma 5: By Lemma 4, we know that either $f_w \leq \frac{1}{2}$ or $f_s \leq \frac{1}{2}$. If $f_w \leq \frac{1}{2}$, then consider the $\frac{1}{2}$ probability where the coin producer is chosen from the PoW nodes.

Define set $\mathbb{X}$ to be such that $\mathbb{X}$ contains the top $|\mathbb{X}|$ PoW solutions in the system (from all malicious and honest nodes) and $\mathbb{X}$ includes exactly m PoW solutions from honest nodes. Except for negligible probability with respect to m, we have $|\mathbb{X}| \leq \frac{5}{4} \times \frac{m}{1-f'_w} < \frac{5m}{2}$. For each $A \in \mathbb{W} \cup \mathbb{S}$, we must have $\mathbb{W}_A \subseteq \mathbb{X}$. Let $\mathbb{Y} = \cup_{A \in (\mathbb{W} \cup \mathbb{S})} \mathbb{W}_A$, and then we have $|\mathbb{Y}| \leq |\mathbb{X}|$. Among all $C \in \mathbb{Y}$, the probability that the C with the smallest $\texttt{hash}(\texttt{beacon}|C\text{'s public key})$ belongs to $\mathbb{W}$ is at least $\frac{(1-f_w)m}{|\mathbb{Y}|}$. Note that $C \in \mathbb{W}$ implies that C is honest. Hence the probability of having an honest PoW node C as the coin producer is at least $\frac{1}{2} \cdot \frac{(1-f_w)m}{|\mathbb{Y}|} \geq \frac{1}{10}$. Such a node C will then send its locally-generated random bit b to all nodes in $\mathbb{W}_C \cup \mathbb{S}_C$, which is a super set of $\mathbb{W} \cup \mathbb{S}$. Then $\texttt{CommonCoin}()$ will return the bit b on all nodes in $\mathbb{W} \cup \mathbb{S}$.

If $f_s \leq \frac{1}{2}$, the lemma can also be proved similarly, while leveraging the fact that $\mathbb{S}_A$ is the same for all $A \in \mathbb{W} \cup \mathbb{S}$. □

D. Proof for Theorem 6

Proof for Theorem 6: We first prove **Validity**. Consider any node $A \in \mathbb{W} \cup \mathbb{S}$. By Theorem 12, we must have $\langle \texttt{grade}_A, \texttt{block}_A \rangle = \langle 2, x_L \rangle$ after Line 9. This implies $x_A = 1$ immediately before Line 11. By Theorem 8, we know that x_A will remain 1 after Line 11. Hence A must return x_L.

We next prove **Agreement**. If every node $A \in \mathbb{W} \cup \mathbb{S}$ has $\texttt{grade}_A \leq 1$ immediately after Line 9, then x_A must be 0 immediately before Line 11 on all A. By Theorem 8, we know that x_A will remain 0 after Line 11. Hence all $A \in \mathbb{W} \cup \mathbb{S}$ must return the empty block, and we are done.

If some node $A \in \mathbb{W} \cup \mathbb{S}$ has $\texttt{grade}_A = 2$ immediately after Line 9, then consider any other node $B \in \mathbb{W} \cup \mathbb{S}$. By Theorem 11, we must have $\texttt{block}_B = \texttt{block}_A$ immediately after Line 9. To prove **Agreement**, it then suffices to prove that $x_B = x_A$ at Line 12: If $x_B = x_A = 0$ for all B, then all nodes in $\mathbb{W} \cup \mathbb{S}$ will return the empty block. If $x_B = x_A = 1$ for all B, then all nodes in $\mathbb{W} \cup \mathbb{S}$ will return $\texttt{block}_A$.

To prove $x_B = x_A$ at Line 12, we focus on the k sequential iterations of $\texttt{Generalized\_GC\_01}()$ at Line 11. For each such iteration, Theorem 7 tells us that there can never be two nodes in $\mathbb{W} \cup \mathbb{S}$ such that one of them returns at Line 24 while the other node returns at Line 25. This then implies that we must have one of the following three (mutually-exclusive) cases:

- All nodes in $\mathbb{W} \cup \mathbb{S}$ return $\texttt{CommonCoin}()$ at Line 26.
- Some nodes in $\mathbb{W} \cup \mathbb{S}$ return 0 at Line 24, and all remaining nodes in $\mathbb{W} \cup \mathbb{S}$ return $\texttt{CommonCoin}()$ at Line 26.
- Some nodes in $\mathbb{W} \cup \mathbb{S}$ return 1 at Line 25, and all remaining nodes in $\mathbb{W} \cup \mathbb{S}$ return $\texttt{CommonCoin}()$ at Line 26.

Let p_0 be the probability that $\texttt{CommonCoin}()$ returns 0 on all nodes in $\mathbb{W} \cup \mathbb{S}$. Similarly define p_1. By Lemma 5, there exists positive constant p, such that $p_0 \geq p$ and $p_1 \geq p$. It is easy to see that in each of the above three cases, with probability at least p, all nodes in $\mathbb{W} \cup \mathbb{S}$ return the same value from $\texttt{Generalized\_GC\_01}()$. We call such an iteration (out of the k iterations) as *successful*. Then except with negligible probability with respect to k (specifically, $(1-p)^k$), there exists at least one successful iteration. Furthermore, if iteration i is successful, then by Theorem 8, all later iterations must be successful as well. Hence except with negligible probability, the last iteration is successful. This then means $x_B = x_A$ for all $B \in \mathbb{W} \cup \mathbb{S}$. □

E. Running $\texttt{Generalized\_GC\_012}()$ *on Block Hashes Rather Than on Blocks*

A simple optimization. All line numbers in this section refer to lines in Algorithm 2. Recall that $\texttt{Generalized\_BA\_Core}()$ invokes $\texttt{Generalized\_GC\_012}()$ at Line 9, which returns a tuple $\langle \texttt{grade}_A, \texttt{block}_A \rangle$. Here $\texttt{block}_A$ is a block. With the optimization, we instead invoke $\texttt{Generalized\_GC\_012}()$ over the hashes of the blocks, rather than over the blocks themselves. This means that with our optimization, $\texttt{Generalized\_GC\_012}()$ at Line 9 will return a tuple $\langle \texttt{grade}_A, \texttt{blockhash}_A \rangle$, where $\texttt{blockhash}_A$ is the hash of some block.

Of course, eventually, we still need a block rather than just a block hash. To do so, after Line 8 chooses a block proposer, if a node is a block proposer according to its own view, it will construct a new block and propagate that block. Such propagation is done before the algorithm continues on to Line 9. We further modify Line 10 as following:

- Old version: "**if** $\texttt{grade}_A = 2$ **then** $x_A \leftarrow 1$ **else** $x_A \leftarrow 0$; **endif**"
- New version: "**if** ($\texttt{grade}_A = 2$ and I have seen the block corresponding to $\texttt{blockhash}_A$) **then** $x_A \leftarrow 1$ **else** $x_A \leftarrow 0$; **endif**"

Finally, after $\texttt{Generalized\_BA}()$ returns a block hash y, if a node has seen the block corresponding to that hash, the node will propagate that block to help other nodes to get that block.

Why this works. To see why this works, first note that the final guarantees of $\texttt{Generalized\_BA\_Core}()$ in Theorem 6 continue to hold. In particular, the **Validity** property will continue to hold because if the block proposer is honest, then it will faithfully propagate the block and the block will reach all nodes before Line 9. Hence the condition of "I have seen the block corresponding to $\texttt{blockhash}_A$" will be satisfied on every honest node.

Second, if $\texttt{Generalized\_BA\_Core}()$ returns $\texttt{blockhash}_A$ on some node $A \in \mathbb{W} \cup \mathbb{S}$, then there must exist some honest node $B \in \mathbb{W} \cup \mathbb{S}$ having $x_B = 1$ immediately after Line 10. The reason is that otherwise, by Theorem 8, x_A would be 0 at Line 12, and $\texttt{Generalized\_BA\_Core}()$ would return the empty block (instead of $\texttt{blockhash}_A$) on A. Now since B has $x_B = 1$ immediately after Line 10, we know that B must have seen the block corresponding to $\texttt{blockhash}_B$, and must have $\texttt{grade}_B = 2$. By Theorem 6, we must have $\texttt{blockhash}_B = \texttt{blockhash}_A$. This means that B holds the block that A needs, and A will get this block once B propagates that block.