Iniva: Inclusive and Incentive-compatible Vote Aggregation

Arian Baloochestani, Hanish Gogada, Leander Jehl, Hein Meling
Department of Electrical Engineering and Computer Science
University of Stavanger
Stavanger, Norway
{arian.masoudbaloochestani, hanish.gogada, leander.jehl, hein.meling}@uis.no

***Abstract*—Many blockchain platforms use committee-based consensus for scalability, finality, and security. In this consensus scheme, a committee decides which blocks get appended to the chain, typically through several voting phases. Platforms typically leverage the committee members' recorded votes to reward, punish, or detect failures. A common approach is to let the block proposer decide which votes to include, opening the door to possible attacks. For example, a malicious proposer can omit votes from targeted committee members, resulting in lost profits and, ultimately, their departure from the system.**

This paper presents Iniva, an inclusive and incentive-compatible vote aggregation scheme that prevents such vote omission attacks. Iniva relies on a tree overlay with carefully selected fallback paths, making it robust against process failures without needing reconfiguration or additional redundancy. Our analysis shows that Iniva significantly reduces the chance to omit individual votes while ensuring that omitting many votes incurs a significant cost. In addition, our experimental results show that Iniva enjoys robustness, scalability, and reasonable throughput.

***Index Terms*—Committee-based blockchains, Vote omission attack, Vote inclusion, Signature aggregation, Incentive-compatible**

I. Introduction

Recently, Ethereum [1], the second largest permissionless blockchain system and the most popular smart-contract platform, completed its shift from Proof-of-Work (PoW) to a Proof-of-Stake (PoS)-based consensus mechanism [2]. Similar to other networks, like Cosmos [3] or Algorand [4], Ethereum now uses a committee-based consensus mechanism. In committee-based consensus, a new block needs to be accepted and voted for by multiple processes from a committee. Committee-based consensus can improve security and finality of PoS [5], [6]. However, committee-based consensus creates new challenges, e.g., how to reward committee members. To encourage participation and prevent free-riding, both Cosmos and Ethereum reward only active committee members [7]. Here, active committee members are detected through the inclusion of their signatures in the blockchain. Therefore, it is crucial that the system includes all active members' signatures in fault-free cases.

This reward scheme introduces the possibility of novel forms of attacks. One such attack is the *vote omission* attack, wherein a malicious actor or a colluding subset of the committee intentionally omits votes from a targeted victim. This can drastically affect the victim's profitability and could even deter them from further participation in the system [8]. While existing systems attempt to mitigate vote omission through carefully crafted incentive mechanisms [3], [9], these strategies fail to address essential concerns. Attacks can occur even when there is no immediate, discernible monetary gain for the attacker. For instance, an attacker could strategically offset their losses through external mechanisms, such as short-selling on another platform. Consequently, relying solely on monetary deterrents may be insufficient for preventing malicious activities like vote omission. A more nuanced approach to incentives is crucial for enhancing the robustness of these systems.

Moreover, vote omission attacks are also feasible in permissioned systems without a reward mechanism. For example, Carousel [10] uses vote inclusion to select processes eligible for leadership. Thus, a vote omission attack in this context may reduce the chances of electing a correct leader.

Addressing the issue of targeted vote omission is challenging. Preventing omissions by individual processes requires redundant aggregation paths. However, existing randomized approaches that use redundant paths allow free-riding. Randomized approaches remain functional even when a large fraction of processes evade their aggregation duties, free-riding on others' work. Vote aggregation, with its compute-intensive signature verification, is particularly attractive to avoid, especially if pairing-based signatures like BLS [11] are used. Such free-riding again reduces redundancy in vote aggregation and thus simplifies vote omission. Hence, we want aggregation protocols that are *incentive-compatible*, meaning that processes face penalties or forfeit rewards if they neglect their aggregation responsibilities.

We analyze existing aggregation schemes. Tree-based protocols like Kauri [12] and ByzCoin [13] lack the necessary redundancy to guard effectively against these attacks. On the other hand, randomized approaches like Handel [14] and Gosig [15] offer redundant aggregation paths but, ironically, this redundancy enables free-riding. Our in-depth analysis shows that Gosig is only effective at mitigating vote omission under specific configurations. Moreover, the very presence of free-riding exacerbates the potency of vote omission attacks.

This paper introduces Iniva, a novel method to aggregate

This work is partially funded by the BBChain and Credence projects under grants 274451 and 288126 from the Research Council of Norway.

2158-3927/24/$31.00 ©2024 IEEE
DOI 10.1109/DSN58291.2024.00050

votes in committee-based blockchains. Instead of relying on incentives, Iniva leverages the properties of indivisible multi-signatures to effectively counteract vote omissions. In common multi-signature schemes like BLS [11], aggregated signatures cannot be decomposed into their constituent parts. Moreover, Iniva organizes processes in a two-level tree. With this structure, the root cannot omit individual votes aggregated at lower levels, while votes omitted at intermediate levels can be re-added. This design effectively neutralizes targeted vote omissions.

Iniva adopts a redundancy model based on fallback paths, activated only when required. This approach strikes a balance, avoiding redundancy in fault-free scenarios while offering robustness against process and link failures. By employing fallback paths, Iniva eliminates the complex reconfiguration steps commonly used in other protocols [12] to find a working tree. Additionally, Iniva's reward mechanism discourages free-riding during vote aggregation. Since fallback paths are activated only under specific conditions, Iniva can precisely determine which processes have fulfilled their aggregation duties.

We integrated Iniva into the HotStuff consensus algorithm [16]. Our experiments show that Iniva ensures vote inclusion, even in the presence of faults. Additionally, Iniva is scalable and has a reasonable performance overhead. In summary, our key contributions are as follows:

- We define *indivisibility* as a property for multi-signature schemes, a property provided by existing aggregation schemes like BLS, and demonstrate its efficacy in mitigating targeted vote omission attacks.
- We present Iniva, a robust vote aggregation and reward scheme for committee-based blockchains that significantly improves security against vote omission attacks.
- We analyze our rewarding scheme using game theory, and prove its incentive-compatibility.
- We analyze Iniva's security and evaluate its effectiveness. Our analysis shows that for an attacker controlling 10% of the processes, the chances to omit an individual signature are reduced by a factor of 10, while the cost of larger exclusion is increased by a factor of 7.
- We simulate vote omission attacks against Gosig and analyze the impact of free-riding.
- We elaborate on the integration of Iniva into the HotStuff protocol and conduct several experiments to analyze Iniva's effectiveness in terms of scalability, throughput, latency, and vote inclusiveness.

II. Background

A. Committee-based Blockchains

Blockchain is a list of blocks cryptographically linked to form a distributed ledger maintained and shared among all participants in a network. Each block contains some data, e.g., transactions detail. In addition, to ensure the integrity of the blockchain, each block also contains the hash of its previous block. Hence, once a block is added to the blockchain, it is considered immutable since any modification to the block would also change its hash. Each process in the network holds a public/private key pair, and their identities are verified through digital signatures.

Processes need to follow a *consensus algorithm* to agree on the inclusion of blocks into the chain. Bitcoin [17] introduced the PoW consensus algorithm. While PoW guarantees security, it suffers from several drawbacks, such as probabilistic consistency (forks) and high computational overhead [18]. To this end, some blockchains [4], [19]–[22] adopt classical Byzantine Fault Tolerance (BFT) protocols [23], [24] as the consensus algorithm. However, as these protocols do not scale to a large number of processes, these methods use a small committee to run the consensus algorithm.

In committee-based blockchains, first, a leader is elected to propose a block. Then, a selected committee verifies the block and votes by signing the block using digital signature schemes. Leaders gather the signed blocks, and if a block gains more than a fraction of the votes, the block is considered approved.

Designing a fair rewarding mechanism for committee-based blockchains is challenging. To prevent free riding, only active members should get rewarded [25]. Most current protocols rely on the leaders to detect the active members by collecting the list of voters. As the leaders might deviate from the protocol, existing methods incentivize them to act correctly. Cosmos [3] introduces the *variational bonus* mechanism in which leaders receive an extra fraction of the reward based on the number of votes they collect from the previous committee. Rebop [9] proposes a reputation-based leader election mechanism with the reputation defined as the number of collected votes in the last T rounds as the leader.

B. Multi-Signature Aggregation

Some committee-based blockchains elect one process as the leader to propose new blocks and receive all the votes to reduce the message complexity. The scalability of these blockchains is dependent on the computational and network capacity of the leader [12]. Some previous works, such as HotStuff [16] rely on multi-signature aggregation schemes to reduce the message size by compacting all signatures into a single signature. However, since HotStuff adopts a star topology, this puts even more load on the leaders by making them responsible for signature aggregation and sharing the result with all committee members. Prior works have proposed decreasing the leader's load by distributing the aggregation work over some or all of the processes. Kauri [12] and ByzCoin [13] use a tree overlay, where parents aggregate their children's votes. Gosig [15] uses a randomized overlay. We discuss these approaches in more detail in the following.

1) HotStuff: HotStuff is a consensus protocol that operates in a sequence of views, each involving three voting rounds. The first round is the *prepare* round, where the leader proposes block B for view v and height h. The committee members then validate B and vote with a *prepare* message if they have not already prepared a block with a higher view at the same height. Once the leader receives enough *prepare* votes, it aggregates them into one signature called a quorum certificate (QC) and shares it with all processes. Processes record the received QC and respond with a *pre-commit* message. The leader waits for enough *pre-commit* replies, forming another QC. This QC is then sent to all processes in the final *commit* round, resulting in a block commitment once enough commit votes are received. According to Yin et al [26], protocols like Casper FFG [27] used in Ethereum, or Tendermint [19] used in Cosmos can be seen as variants of HotStuff.

To achieve better performance, the three rounds can be performed concurrently for three different views. This variant is called chained HotStuff, where a single QC can serve as *prepareQC*, *pre-commitQC*, and *commitQC* at the same time.

HotStuff, can use different leader election policies. Blockchains typically adopt the Leader-Speak-Once (LSO) model [28], [29], where every leader only proposes a single block and the leader is changed every view. LSO minimizes the leader's power over new block proposals and makes the protocol more fair.

2) Tree-based approaches: Kauri and ByzCoin use a tree for distributing the signature aggregation work among the processes. The tree-based topology reduces the workload on the leaders compared to the HotStuff star topology because each parent process is responsible for aggregating its sub-tree. In case of failure, these protocols require reconfigurations and may fall back to a star topology in cases with many failures [12]. Kauri proposes a reconfiguration mechanism for trees with height 2. In these trees, processes need to aggregate $\mathcal{O}(\sqrt{n})$ many signatures. Kauri uses pipelining techniques to achieve high throughput despite the added latency through communication on the tree. However, while the aggregation work is distributed among the processes, the parent processes have complete control over their sub-tree and are able to exclude leaf children from the aggregated signature.

3) Gosig: Gosig [15] is a BFT protocol for committee-based blockchains. In Gosig, leaders are selected secretly using a Verifiable Random Function (VRF) and share their block proposals with other processes. Each process performs signature aggregation and repeatedly shares its current aggregate with k other processes, selected at random from the complete committee.

III. System Model

A set $\Pi = \{p_1, p_2, ..., p_n\}$ of processes are available in the committee. For simplicity, we assume Π to be constant and do not consider the committee selection protocols. The fixed membership assumption is to simplify explanation and analysis. Our solution also works for dynamic committees as long as committee members for one view are known a priori. We assume a synchronous network with an upper bound Δ on the delivery of any message between correct participants. Our system requires synchrony to ensure inclusiveness. In the case of an eventually synchronous system, it ensures inclusiveness after global stabilization time [30].

We assume an adversary controlling a fraction m of the processes in the committee, where $m \leq f = 1/3$. Processes under the control of the adversary may behave arbitrarily. However, we are especially interested in the case where the adversary tries to diminish the reward received by one victim $p_v \in \Pi$. We assume that the adversary cannot disturb the processing and communication between correct processes. Thus, denial of service attacks are out of scope.

We assume each process p_i in the system has a private/public key pair sk_i/pk_i and access to a list of other processes with their public keys.

A multi-signature scheme is a digital signature scheme that allows the aggregation of signatures. Let $\sigma_1 = \text{sign}(m, sk_1)$ and $\sigma_2 = \text{sign}(m, sk_2)$ be signatures for a message m produced with different private keys. The signatures can be aggregated with multiplicity i and j where $i, j \in \mathbb{Z}$:

$$\sigma' = \text{agg}(\sigma_1^i, \sigma_2^j)$$

The resulting signature σ' can be verified by aggregating the corresponding public signatures with the same multiplicity:

$$\text{verify}(\sigma', pk_1^i pk_2^j)$$

We assume processes have access to an *indivisible multi-signature* scheme, such that given σ', it is infeasible to retrieve σ_1 or σ_2. For pairing-based signatures, indivisibility of up to k signatures was proposed as an assumption by Boneh et al [31]. We use BLS signatures [32], which are indivisible according to Coron and Naccache [33].

At the beginning of each round, p_i is assigned a unique ID ($ID[p_i] = i$). We assume the processes have access to a deterministic shuffling algorithm, and Π is shuffled every round so that the IDs will be different at each round of the protocol. The shuffling algorithm needs to be unpredictable, meaning that the processes cannot predict the outcome of the shuffling for future rounds. As an example, the above algorithm can be implemented using a VRF [34].

IV. Problem Statement

In committee-based blockchains a leader disseminates a block to participants, who return votes/signatures to the leader. The leader then outputs an aggregate of these votes, aka a QC. We present a slight variation of Kauri's [12] vote aggregation scheme below.

Definition 1. A *vote aggregation scheme* has an interface with the following communication primitives:

- **broadcast**(B). Invoked by the leader to disseminate a block B and start vote aggregation.
- Upcall *deliver*(B) at p_i delivers B. p_i emits a vote for block B:
$$\textit{vote}(B) = \begin{cases} \sigma_{B,i} & \text{if } B \text{ is valid} \\ \perp & \text{if } B \text{ is invalid} \end{cases}$$
- Upcall *aggregate*(B, QC_B, md) at the leader delivers an aggregate QC_B of valid signatures from *vote* and additional metadata md specifying which processes' votes are included.

Neiheiser et al [12] define the following liveness properties for a vote aggregation scheme:

Definition 2 (*Reliable Dissemination*)**.** If the leader is correct, all correct processes deliver the block sent by the leader.

Definition 3 (*Fulfillment*)**.** If the leader is correct and all correct processes invoke vote with a valid signature, then the leader emits a QC containing at least $(1-f)N$ signatures.

These properties are sufficient to ensure liveness and safety of HotStuff [12]. Additionally, straightforward validity properties are expected, i.e. that correct processes only deliver blocks actually sent by the leader, and that QC_B only includes valid signatures.

In this work we are interested in the LSO model, where the leader changes after every block. We therefore adapt the vote aggregation scheme, assuming that **broadcast** is invoked by the leader proposing B, while *aggregate* happens at the next leader. Further, we require reliable dissemination and fulfillment to hold only if two consecutive leaders are correct.

A. Rewarding

Some committee-based cryptocurrencies use the QC to reward participants. For example in cryptocurrencies like Cosmos [3], Solidus [35], or Ethereum [1], the QC is used to detect active committee members and reward them accordingly to prevent free riding.

Such rewarding schemes can be modelled as a function *reward*(QC), which computes a distribution of rewards based on the quorum certificate. Since the QC is included in the next block, the reward distribution can be verified by every process, re-computing the *reward* function.

Inclusiveness: If the QC is used for rewarding, it is crucial for these methods to guarantee the inclusion of all non-faulty processes within the QC. We refer to this attribute as being *inclusive*.

Definition 4 (*Inclusiveness*)**.** If the current and next leader are correct, then all signatures from correct processes are contained in the aggregated QC.

We note that Inclusiveness may also be useful in other contexts. For example, Carousel [10] proposes a reputation-based leader rotation mechanism that looks at the previous QCs to avoid selecting failed processes as leaders. Using Carousel, inclusiveness can guarantee that all correct processes actually can become the leader.

B. Vote omission

Since leaders are in charge of forming QCs, a malicious leader can ignore some of the votes and form the QC with the processes it desires. We refer to this attack as the *vote omission* attack. Incentive engineering [9] can ensure vote omissions are not profitable. However, attacks are still possible. Especially attacks targeted at an individual process may have a devastating effect on the victim, while only incuring a small cost to the attacker. In *targeted vote omission*, an attacker controlling a large fraction of the committee tries to omit as many votes from a specific process as possible. In these attacks, the attacker does not intentionally omit other processes unless it leads to a more successful attack. We define *collateral* as the number of non-target processes that an attacker is willing to exclude to perform the attack. For example, with a collateral of 0 only the target will be excluded and no other processes. To measure the robustness of a protocol against targeted vote omission attacks, we define *c-omission probability*.

Definition 5 (*c-omission probability*)**.** We define the c-omission probability as the probability for an attacker to successfully perform a targeted vote omission attack with collateral at most c during one instance of vote aggregation based on a random assignment of processes to the attacker and the victim role. The probability space is the set of all possible process assignments. We assume all such assignments to be equally likely. Omission probability is a function in $m \in [0,1]$, the fraction of the committee's processes controlled by the attacker.

For instance, the HotStuff protocol adopts a round-robin leader selection scheme. Thus, an attacker controlling a fraction m of the processes can become the leader m fraction of the time. Given that each leader has the authority to decide which votes to incorporate, the probability of the attacker executing the targeted vote omission attack is m.

We note that as an attack probability, a c-omission probability of m^2 signifies a more robust protocol, than c-omission probability of m.

C. Free riding

Vote aggregation schemes that support redundant aggregation are susceptible to free riding. Free riding by other processes (neither victim, nor attacker) helps an attacker to perform vote omission. For example, in Gosig, all processes are expected to participate in vote aggregation. However, some processes may decide to omit the aggregation step to avoid costly signature verification, and instead, only disseminate their own signature. If other processes follow this *free riding* behavior, it simplifies a targeted vote omission of correct processes, as our simulations show (see Section VII). To avoid such free riding, we require vote aggregation to be *incentive compatible*.

Definition 6 (*Incentive compatibility*)**.** A rewarding scheme is incentive compatible if following the protocol gives higher utility compare to other strategies.

D. Alternative approaches

While existing approaches for signature aggregation also use indivisible multi-signatures, they have multiple shortcomings. A summary of the existing protocols' drawbacks is shown in Table I.

Existing tree-based signature aggregation approaches such as Kauri or ByzCoin fail to prevent vote omission attacks as the internal processes in the tree have direct control over their children and are able to selectively omit them. Both Kauri and ByzCoin use a stable tree whose reconfiguration is triggered by the leader. This allows an attacker in charge of the leader

TABLE I: A comparison between existing multi-signature aggregated schemes

	0-omission probability	Inclusive	Incentive compatible
Star protocol	m	Yes	Yes
Randomized tree	m[a]	No	Yes
Gosig (k)	k-dependent[b]	No	No
Iniva	m^2	Yes	Yes

[a] In a static configuration, the leader may perform the attack every round.
[b] The 0-omission probability of Gosig depends on k. See Section VII.

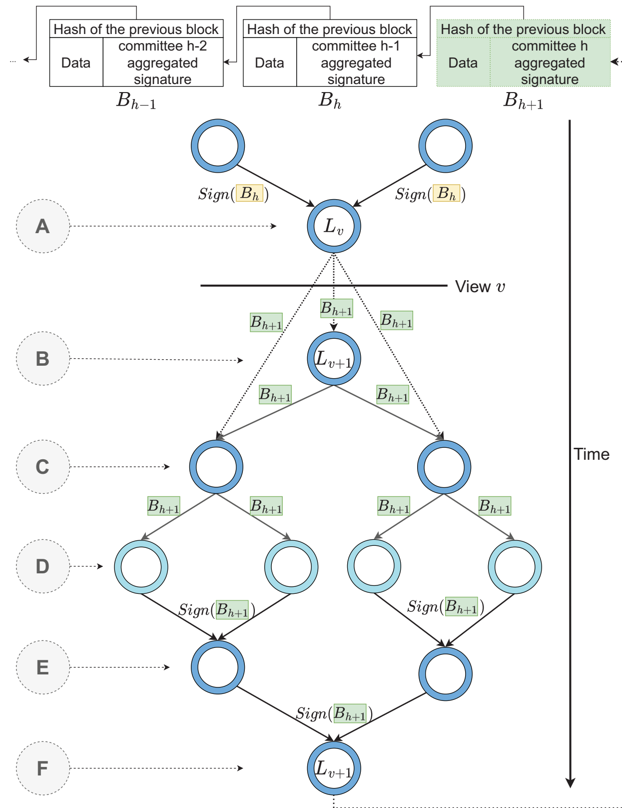


Fig. 1: An overview of Iniva. **A)** L_v commits B_h. It creates and forwards B_{h+1} to L_{v+1} and L_{v+1} children. **B)** L_{v+1} receives B_{h+1} and starts the view by sharing the proposal with its children. **C)** Internal nodes forward B_{h+1} to their children, and wait for their response. **D)** Leaf nodes verify and sign B_{h+1}, and share their signature with their parent. **E)** Internal nodes aggregate their children signatures, and share it with their parent. **F)** L_{v+1} commits B_{h+1}. It creates and forwards B_{h+2} to L_{v+2} and L_{v+2} children.

to arrange a configuration where it also controls the parent of the victim. Additionally, the failure of internal processes leads to the loss of the whole sub-tree under them. This can result in omissions even in the absence of attacks since these methods are not inclusive. Complex reconfiguration is needed in case of failures to rearrange the tree.

Gosig uses a randomized, redundant communication pattern for vote aggregation. The inclusion of a given process in the QC is therefore probabilistic, even in fault-free cases. Here, if the attacker receives the victim's individual signature early in the aggregation process, it will be able to remove it from the final certificate. We performed simulations on the omission probability of Gosig, which shows that it can reduce targeted vote omissions only for small values of k and attackers controlling only a small fraction m. For larger values, Gosig 0-omission probability is m, allowing targeted omission every time the attacker is selected as leader. Additionally, Gosig is vulnerable to free-riding, which simplifies targeted vote omission.

Another approach to reduce vote omission is to let processes compete in aggregation and use the process aggregating the most signatures as the next leader. A similar approach was applied in Rebop [9]. Unfortunately, this approach opens novel attacks. An attacker may hold back its own signature, thus reducing others' chances of leadership. Note that as incentive engineering and reputation-based schemes such as Rebop [9] can defend against targeted vote omission attacks with large collateral, we are mostly interested in collateral of 0.

In the next section, we show how Iniva avoids reconfiguration and omission using a tree-based overlay and its extension with an incentive scheme that prevents free riding.

V. Iniva

In committee-based blockchains, committee members work together to append a new block to the blockchain through several views. The current length of the blockchain is represented through the parameter height h. Processes move to the next view if they fail to append a new block, while height remains unchanged. At each view v, one of the processes is selected as the leader ($L_v \in \Pi$) and is responsible for proposing a new block. For adding the proposed block to the blockchain, L_v must gather at least $1 - f$ fraction of the votes from the previous committee, where f defines the maximum fraction of faulty processes that the protocol can handle (e.g., $f = 1/3$). An aggregated signature of $1 - f$ fraction of the committee is called a QC. The QC of the last approved block is called the highest QC. L_v uses the highest QC to distribute a reward R among the members whose votes are included.

In this section, we present Iniva, an Inclusive and Incentive Compatible Vote Aggregation mechanism in committee-based blockchains. In the following, we first discuss the proposal propagation and vote aggregation in Iniva, and then we present a rewarding scheme that makes Iniva incentive compatible.

A. Signature Aggregation

In this section we discuss the block propagation and signature aggregation procedures in Iniva, which are shown in Algorithm 1 and Figure 1.

At the start of each view v, the leader of that view, L_v creates a new block extending the blockchain at current height h, B_{h+1}. Based on the QC and view number included in the block, all processes generate the same tree for the given view (Lines 4-5, Line 8). L_v then forwards the block to the

Algorithm 1 Block propagation and signature aggregation

1: **Process Variables:**
2:     *parent* $\triangleright$ Direct parent of the process in the tree
3:     *aggSig* $\triangleright$ The aggregated signature
4: **on broadcast**(B) $\triangleright$ at leader L_v
5:     $root, children \leftarrow$ makeTree(B)
6:     **send** $\langle \text{PROPOSAL}, B \rangle$ **to** *root* and *children*
7: **on** $\langle \text{PROPOSAL}, B \rangle$
8:     *parent, children* $\leftarrow$ makeTree(B)
9:     **if** $children \neq \emptyset$ **then**
10:         **send** $\langle \text{PROPOSAL}, B \rangle$ **to** *children*
11:     $deliver(B)$
12:     $\sigma_B \leftarrow vote(B)$
13:     $aggSig \leftarrow aggSig \cup \sigma_B$
14:     **if** $children = \emptyset$ **then**
15:         **start** *aggTimer*
16:     **else** $\triangleright$ tree leaf
17:         **send** $\langle \text{SIGNATURE}, \sigma_B \rangle$ **to** *parent*
18: **on** $\langle \text{SIGNATURE}, sig \rangle$
19:     **assert** verifies$(sig, sig.signers)$
20:     $aggSig \leftarrow aggSig \cup sig$
21: **on** timeout(*aggTimer*)
22:     **if** isRoot(*self*) **then** $\triangleright$ $root$ is L_{v+1}
23:         $missing \leftarrow \Pi - aggSig.signers$
24:         **send** $\langle \text{2ND-CHANCE}, B \rangle$ **to** *missing*
25:         **start** *secondChanceTimer*
26:     **else**
27:         **send** $\langle \text{SIGNATURE}, aggSig \rangle$ **to** *parent*
28:         **send** $\langle \text{ACK}, aggSig \rangle$ **to** *children*
29: **on** $\langle \text{ACK}, sig \rangle$
30:     **assert** verifies(sig)
31:     $aggSig \leftarrow sig$
32: **on** $\langle \text{2ND-CHANCE}, B, proof \rangle$ from p
33:     **assert** isValid$(B, proof, p)$
34:     **if** B has new view **then**
35:         $deliver(B)$
36:         $\sigma_B \leftarrow vote(B)$
37:         $aggSig \leftarrow aggSig \cup \sigma_B$
38:     **send** $\langle \text{SIGNATURE}, aggSig \rangle$ **to** *sender*
39: **on** timeout(*secondChanceTimer*) $\triangleright$ at L_{v+1}
40:     $aggregate(aggSig, aggSig.signers)$

root process in the tree and its children (Line 6, Figure 1-A). After receiving and verifying a block, a process builds the tree itself and forwards the block to its children. Processes without children (tree leaves) instead send their signatures to their parents (Lines 7-17).

Each internal process in the tree verifies and aggregates the received signatures together with its own signature (Lines 18-20). Upon a timeout, or once aggregation for all children is completed, the process forwards the aggregated signature to its parent (Line 27). It also sends an acknowledgement (*ack*) to its children (Line 28). The ack includes the aggregated signature and acts as proof that the parent has included the signatures of the senders.

Due to network issues or malicious processes in the tree, some processes may not receive the proposal and aggregated signatures may be incomplete. The root process in the tree is the leader of the next view L_{v+1}. The root process collects the signatures to a QC, which it uses to create the next block. Before creating the next block, L_{v+1} gives one last chance to the processes whose votes are not included by sending them a 2ND-CHANCE message. L_{v+1} does send this message either once a QC has been collected or upon a timeout (Lines 22-25).

Replying to a 2ND-CHANCE message with their individual signature enables the message sender to exclude a process. Therefore, processes reply to a 2ND-CHANCE with the aggregated signature received from their parent in an *ack* message. Otherwise, 2ND-CHANCE messages are validated according to function isValid. A second chance message is valid if it includes a quorum of signatures, or a signature from the parent, but not the current process's signature. Additionally, a second chance message may also be valid if sufficient time has passed since the block creation. This can be checked by comparing the block timestamp against the current time.

Since the internal tree processes do more work than other processes, we propose a mechanism to reward them for their extra work.

B. Rewarding Mechanism

We now explain our rewarding mechanism. Rewards are distributed by the leader or root. We first explain how rewards are distributed and then how other processes verify the distribution determined by the leader. We identify the following requirements for our rewarding system:

1) All active committee members should be rewarded.
2) Processes with extra responsibilities, like the internal processes and the leader, should receive an additional reward.
3) Omission of any assigned duties, i.e. voting, aggregation, or 2ND-CHANCE messages, should result in reduced rewards.
4) The total reward paid out per block should be independent of how many votes were aggregated.

We note that requirements 1-3 ensure that processes are motivated to conduct their assigned tasks. Requirement 4 ensures that the aggregation and rewarding procedures do not affect the amount being distributed. This allows, for example, to use fees received from users to be redistributed as a reward. In case rewards are newly minted tokens, this ensures a constant and predictable creation rate. Finally, this also ensures that our rewarding method is not susceptible to attacks, where a process may forfeit some of its rewards but receives a larger fraction of the total reward paid. Such attacks exist in other schemes, e.g. selfish mining [36].

According to Requirement 2 and 1, we use a certain fraction of the total reward to give a bonus for aggregating processes

(b_a), and the leader (b_l) and distribute the remaining reward evenly among all processes, whose signature is included in the final vote $b_v = (1 - b_l - b_a)$.

Let R denote the total reward given out for one block. Due to Requirement 4, the bonus for aggregation and leader is given as a fraction of R. As a bonus for aggregation, internal processes receive $\frac{b_a}{n}R$ for each signature of a child. Similarly, the leader, or root of the tree, receives $\frac{b_a}{n}R$ for each subtree that it aggregates.

For the leader bonus, we use a similar approach as the variational bonus introduced in Cosmos [3], where the leader receives a bonus of $\frac{b_l}{fN}R$ for each signature included in the final certificate, exceeding the minimal requirement of $(1-f)N$ signatures.

The reason for having a separate bonus for the leader is that the leader is the only process that can send 2ND-CHANCE messages to every other process. Therefore, by tying the leader bonus to the number of included processes, we motivate the leader to send 2ND-CHANCE messages to all missing processes.

Finally, we want leaf processes to be aggregated by their parents rather than through 2ND-CHANCE messages. If a leaf process is included via a 2ND-CHANCE message, its parent loses the $\frac{b_a}{n}R$ aggregation bonus. In these cases, we also reduce the voting reward received by the child by $\frac{b_a}{n}R$.

Finally, all remaining reward, after deducing aggregation and leader bonuses and applying punishment for 2ND-CHANCE, is distributed evenly among all the processes in the committee.

We note that to compute the rewards, it is necessary to know who the leader was, which signatures have been included, who performed how many aggregations, and whether signatures have been collected through aggregation or via 2ND-CHANCE messages.

Since the leader and tree can be recreated deterministically, the main issue is determining if a signature has been collected through 2ND-CHANCE messages. For this purpose, we use the fact that the same signatures can also be aggregated multiple times in an indivisible aggregation scheme. Thus, when an internal process aggregates its children, it includes each child's signature twice, while a leader aggregating 2ND-CHANCE messages will include signatures only once. Additionally, the internal process will include its own signature one additional time for each aggregated child.

For example, if a process collects 2 signatures σ_1 and σ_2, it adds its own signature σ_i 2 additional times, resulting in an aggregated signature:

$$aggSig = \text{agg}(\sigma_1^2, \sigma_2^2, \sigma_i^3) \tag{1}$$

The leader does check these multiplicities and only includes correctly aggregated shares. We note that if an internal process or a leaf sets a wrong multiplicity on its signature, this can be detected by the leader. Further, the leader cannot change the multiplicity of signatures reported by internal processes since these are indivisible. To check that aggregation bonuses and 2ND-CHANCE punishments are computed correctly, processes simply compare the multiplicities of the signatures of leaf and internal processes. The leader is considered faulty if the multiplicities reported in a block are wrong.

C. Discussion

Iniva uses a tree-based structure and indivisible multi-signature aggregation scheme to remain inclusive and prevent vote omission attacks. In the absence of failures and attacks, Iniva requires only one tree aggregation, which is comparable to existing tree-based aggregation schemes [12] in terms of latency and throughput. In the presence of partial failures, Iniva relies on fallback paths for fault tolerance.

Theorem 1. Algorithm 1 guarantees Reliable Dissemination.

Proof. According to Definition 2 and our adjustment to LSO, we assume that the leader L_v and the next leader L_{v+1} are correct. L_{v+1} is also the root of the tree used for dissemination. If any correct process p_i does not receive the block through the tree dissemination (Line 11 of Alg. 1), p_i will not send a signature. Therefore L_{v+1} will send a 2ND-CHANCE message to p_i and p_i will deliver executing (Line 35). □

In Iniva we use a tree of height 2 (Algorithm 1). A tree with more levels could provide better protection against vote omission, as the internal processes would also send 2ND-CHANCE messages. However, multiple rounds of 2ND-CHANCE messages, and additional levels would significantly increase latency.

Iniva's maximum latency for each round is 7Δ. Since Δ is the upper bound for message delivery between correct processes, it takes 1Δ for L_{v-1} to share a new block with L_v. Thus, leaf processes receive the block 2Δ later, and it takes another 2Δ for the leader to receive the aggregated messages. Finally, if there are any missing signatures, another 2Δ is added to the overall latency due to the 2ND-CHANCE messages.

Theorem 2. Algorithm 1 guarantees Inclusiveness after 7Δ.

Proof. Let p_i be a correct process whose signature was not received by the root during tree aggregation. Since we can assume that the root and next leader is correct, p_i will receive a 2ND-CHANCE message and reply either with its own signature, or an aggregate received in ACK. In the later case, the aggregate also includes p_i's signature. This signature will be aggregated by the leader. The delay of 7Δ follows from the argument above. □

The following Corollary follows easily, since Inclusiveness actually implies Fulfillment.

Corollary 1. Iniva guarantees Fulfillment.

Note that the number of included votes is also dependent on when the leader send the 2ND-CHANCE messages. Processes that have not received the block from their parents need some time to verify and sign the block. If the leader sends the 2ND-CHANCE within a certain timeout, missing processes have more time to keep up. However, some processes might receive the 2ND-CHANCE message before the acknowledgment from their parent. While increasing timeouts alleviates this problem, it leads to higher latency and lower throughput. Our evaluations (section VIII) show that in presence of failures, lower timeouts result in increased throughput, while larger timeouts favor inclusiveness.

VI. Incentive Analysis

We use game theory to analyze the possible strategies for processes in different roles. We model the system as a two-player game, where each player controls a fraction of the processes. We show that if the player controlling the majority of processes acts honestly, then strategies available to the minority player are dominated by the honest strategy.

a) Player Set: We assume two players, an honest player p_h and an attacker p_a. We assume that p_a controls a fraction $m < 0.5$ of all processes.

b) Strategy Set: The strategies available to players are expressed as $S(e_l, e_v, e_a, e_p)$. The parameters e_l, e_v, e_a, and e_p express different possible attacks. We omit some strategies that are obviously not beneficial. For example, not proposing a block since it results in zero reward. The strategy $S_0 = S(0, 0, 0, 0)$ corresponds to correct behavior. The attacks available to a player depend on its processes' roles in a round: round leader, internal process, and leaf process.

The leader collects signatures for the block. It can submit complete subtrees or individual, 2ND-CHANCE messages from the block. Parameter e_l describes a strategy in which the player tries to omit $e_l \cdot n$ many signatures belonging to the other player. To form a valid block, $e_l \leq f$ must hold.

If a player controls processes that are not the leader, these processes can refrain from voting for a block. We assume $e_v \cdot n$ many processes belonging to the player omit their votes.

Internal processes aggregate signatures in their subtree. They may omit aggregating these signatures, leaving signatures to be aggregated by 2ND-CHANCE messages instead. The player omits aggregation of $e_a \cdot n$ many signatures from processes belonging to the other player.

Leaf processes can refrain from sending their signatures to their parent, sending them in a 2ND-CHANCE message to the leader instead. We assume $e_s \cdot n$ many processes under a player's control do this.

c) Utility Function: We define the player's utility function as its payoff in each round. This payoff includes both the voting reward and the aggregation bonus.

In the following, we analyze the profitability of different strategies for player p_a, assuming that p_h follows S_0. In any strategy S' other than S_0, both p_a looses some rewards compared to S_0. Let $L[S']$ be this loss. The total rewards lost by p_a and p_h ($R[S']$) are redistributed, and p_a gains $m \cdot R[S']$. We derive conditions, such that $m \cdot R[S'] < L[S']$, which ensures S' is dominated by S_0.

A. Vote Omission

A player controlling the leader may omit entire subtrees. In $S(e_l, 0, 0, 0)$ the leader omits $e_l \cdot n$ many votes, belonging to another player. In this case, the voting reward of omitted processes $e_l b_v R$ and the aggregation reward $e_l b_a R$ for these votes are redistributed among all processes. Similarly, the leader bonus is reduced by $\frac{e_l}{f} b_l R$ and redistributed.

With this strategy, player p_a loses at least $\frac{e_l}{f} b_l R$ but gains a fraction m of the redistributed rewards. We deduce the following condition:

$$\frac{e_l}{f} b_l > m \left(\frac{e_l}{f} b_l + e_l b_a + e_l b_v \right) \tag{2}$$

$$\Leftrightarrow b_l > \frac{mf}{1 - m + mf} \tag{3}$$

B. Vote Denial

If a player is in control of non-leader processes, these may refrain from voting. In strategy $S(0, e_v, 0, 0)$ a player refrains from voting with $e_v n$ many of its processes. We only consider this *vote denial* attack when the player does not hold the leader. In this case, the player loses the voting reward for omitted votes $e_v \cdot b_v R$ but gains fraction m of the redistributed leader bonus $\frac{e_v}{f} b_l R$ and aggregation bonus $e_v b_a R$. The lost voting reward is also redistributed. We deduce the following condition:

$$e_v b_v > m(\frac{e_v}{f} b_l + e_v b_a + e_v b_v) \tag{4}$$

$$\Leftrightarrow b_l < \frac{f(1 - b_a - m)}{m + f - mf} \tag{5}$$

C. Aggregation Denial

A leaf process in the tree can not send its vote to its parent and reply to 2ND-CHANCE messages instead. We refer to this attack as *aggregation denial*. We use the parameter e_a for a strategy where $e_a n$ many processes from the player perform this attack. In this attack, the attacker is punished, losing $e_a b_a R$ of its voting reward. This punishment and the denied aggregation bonus $e_a b_a R$ are redistributed. Thus, this attack is not profitable if the following equation holds:

$$m 2 e_a b_a < e_a b_a \tag{6}$$

D. Aggregation Omission

If a player controls an internal process, it can skip aggregating some connected leaf processes, leaving the leaf processes' votes to be collected via 2ND-CHANCE messages. This will result in punishment for the leaf processes. We refer to this attack as *aggregation omission*. If $e_p n$ many signatures from the leaf processes belonging to other players are not aggregated, the attacker loses $e_p b_a R$ of its aggregation reward. The punishment and lost aggregation bonus are redistributed. This results again in Equation 6. For $m < 0.5$, Equation 6 holds and we get the following Lemma:

Theorem 3. For a player p_a with $m < 0.5$, if Equations 3 and 5 hold, then all strategies $S(e_l, e_v, e_a, e_p)$ are dominated by $S(0, 0, 0, 0)$.

Proof. This follows from the analysis above, since the redistributed and lost rewards ($R[S']$ and $L[S']$) for different attacks sum up. □

VII. Security Analysis

This section analyzes the security of Iniva against possible attack scenarios.

A. Targeted Vote Omission

Here we analyze the security of Iniva against targeted vote omission attack with collateral 0, in which the attacker tries to omit an individual vote.

In Iniva, the direct parent is not able to omit its children since the 2ND-CHANCE messages help an omitted process to get re-added by the tree root. Additionally, due to the indivisible multi-aggregation schemes, the root is not able to retrieve and omit one specific signature from the aggregated signatures it receives. Therefore, in order for the attack to be successful, the attacker needs to control two specific processes. If the victim is a tree leaf, the attacker can omit its signature if it controls both the root of the tree and the direct parent of its victim in one view. Considering m to denote the attacker's power as the fraction of committee members the attacker controls, and P is the probability of the victim to be a leaf, the probability of such an attack is $P \cdot m^2$.

Omitting an individual vote is also possible if the victim is an internal process, and the attacker controls both the current and previous view leaders. In this way, the attacker can skip sharing the block proposal with the victim, and collect the victim's children through 2ND-CHANCE messages. Note that controlling both leaders is required for this scenario since the block proposal is created by the leader of the previous view and is shared with both the current view leader and its children. The probability of such a scenario is $(1-P) \cdot m^2$.

Theorem 4. In Iniva, the probability for an attacker with power m, to omit only its target is m^2.

Proof. This is an immediate result of summing the above probabilities: $P \cdot m^2 + (1-P) \cdot m^2 = m^2$ □

Corollary 2. Considering the two attacks above, 0-omission probability of Iniva is m^2.

Note that if an internal process does not respond with an acknowledgment to the received signatures, a process might be lured into replying to a 2ND-CHANCE sent by a faulty leader and gets omitted. Therefore, Theorem 4 holds if the victim receives the acknowledgment from a correct parent before a potential 2ND-CHANCE from the attacker.

An attacker can still exclude a whole branch ($a+1$ processes, considering a leaves for the aggregator) to omit one targeted process by having access to L_v (collateral of a). This is further analyzed in our simulations below. However, existing incentive-based solutions are well suited to prevent such large omissions and may be applied additionally to Iniva.

B. Simulations

To prove the security of Iniva against the mentioned attacks, we conducted different simulations. We use Gosig and a simple star protocol with round-robin leader election as the baseline. Unless mentioned otherwise, in all simulations related to Iniva there are 111 processes in the committee, forming a 2-level tree with fan-out of 10. Results of the simulations are shown in Figure 2.

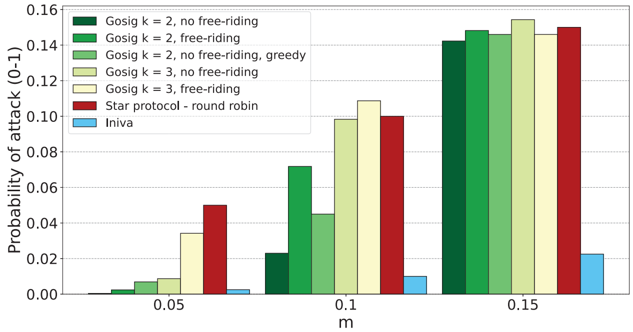
We first simulated the targeted vote omission attack with collateral of 0 in Gosig under different k and different attacking power m. We also looked into situations where 30% of the processes are free riding and also situations where the malicious leader tries to be greedy, and initiates the aggregation process by first sharing the signature with the victims. As shown in Figure 2a, while Gosig can defend against the attack under small k and m, increasing these parameters highers the omission probability of Gosig that of a star protocol. The results also show that free riding makes the attack more successful. For example, while having $k = 2$ and $m = 5\%$ the attack in Gosig happens only 4% of the time, free-riding increases the chances of the attack up to 24%.

In the second simulation we analyzed the robustness of Iniva and Gosig against vote omission attack under different collateral. Figure 2b shows the number of successful omissions based on the collateral. In this simulation, the attacking power m is set to 5%. Different than Gosig, collateral has little effect on omission probability in Iniva, as long as it is not enough to allow removal of a complete sub-tree. Thus Iniva has a reasonable and mostly better omission probability compared to baseline methods under different collateral.

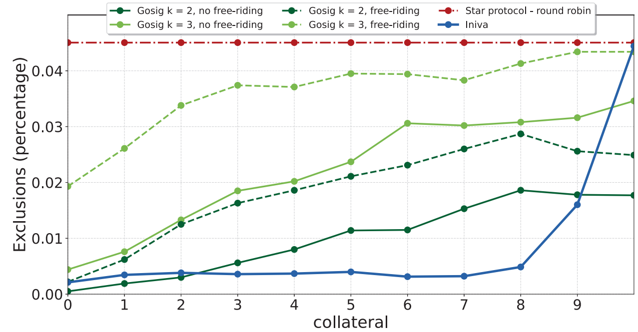
The third simulation compares the fraction of the reward lost by victim and attacker under different attacks in Iniva, with the star protocol as the baseline. In Iniva, we use b_l as 15%, and b_a as 2%. The baseline also uses the same leader bonus, but not aggregation reward.

Figure 2c shows the difference between the reward gained by the victim and attackers with their expected share $(1/111)$. We can see that while in baseline, an attacker with $m = 0.3$ is able to lower the expected share of the victim by vote omission attack almost 25%, in Iniva this is reduced to around 7%. The effect of vote denial attack is almost the same in both baseline and Iniva, but it's is a much more expensive attack compare to vote omission, since the attackers lose much more for performing the attack. We note that, while for a larger attacker, the fraction of reward lost in the attack is reduced, the actual cost still increases.

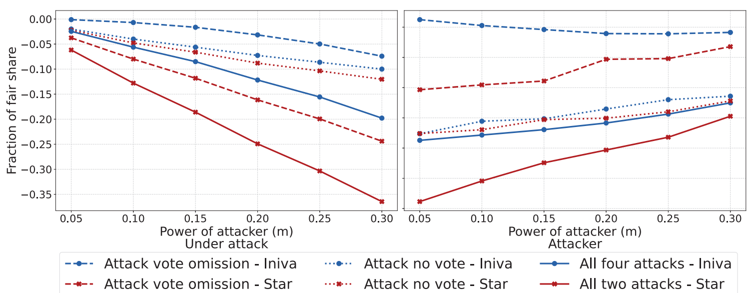
In the fourth simulation we show the effect of the tree configuration (number of internal processes) on vote omission with any collateral. Figure 2d compares how much reward (percentage of the block reward) attacker and victim lose in Iniva having 4 and 10 internal processes (111 and 109 processes in total respectively), and star protocol as the baseline. For example, an attacker with $m = 0.1$ loses 7 times more in Iniva with 10 internal processes compared to the star protocol. Having larger sub-trees makes the attack with high collateral even more expensive due to the larger number of children under each aggregator. We see that an attacker with $m = 0.1$ loses 15 times more in Iniva with 4 internal processes compared to the baseline. This shows while Iniva is unable to reduce the probability of the attack for higher collateral, it effectively increases the cost of the attack, making it more



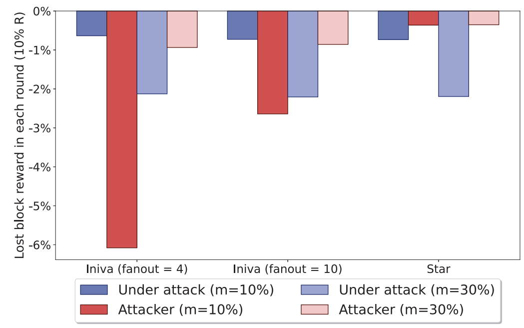
(a) Vote omission probability with collateral 0: This simulation compares the possibility of targeted vote omission attack in Gosig with Star protocol and Iniva under different values of k and m.



(b) Vote omission probability with larger collateral: Percentage of blocks with successful vote omission attack in Iniva, Gosig, and Star protocol with different collateral. m is set to 5%.



(c) Effect of different attacks with collateral 0: It compares the fraction of the fair reward lost by the victim and attacker under different attacks under different (m) for a star protocol with leader reward and Iniva.



(d) Vote omission effect with large collateral: Compares the reward lost (a percentage of the block reward) for both the attacker and the victim when the attacker removes up to a whole branch for omitting its target in Iniva with 4 and 10 internal nodes and the star protocol.

Fig. 2: Simulation results. In each simulation, there are 111 processes in each committee for Iniva (a full 2-level tree with a fan-out of 10). In (a) and (b) there are 100 processes in the committee for Gosig. In (d), there are 109 processes when having 4 internal nodes.

difficult to perform.

VIII. Experimental Results

A. Implementation

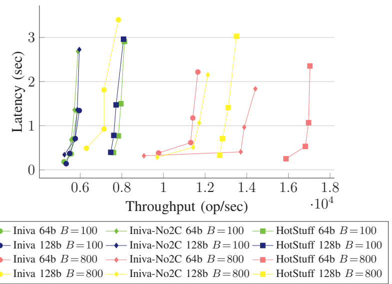
We implemented Iniva, integrating the signature aggregation described in Algorithm 1 in an existing implementation of the HotStuff consensus algorithm [37] [1]. Iniva is added as a module in the framework to perform propagation of blocks and vote aggregation. Iniva does not change the implementation of consensus, or client and request handling.

The HotStuff algorithm operates in synchronous rounds [16]. A new block is only proposed after the votes for the previous block have been aggregated. In this setting, additional latency during dissemination and waiting for additional votes affects not only latency but also the throughput of the protocol. This allows us to realistically evaluate the overhead added by Iniva.

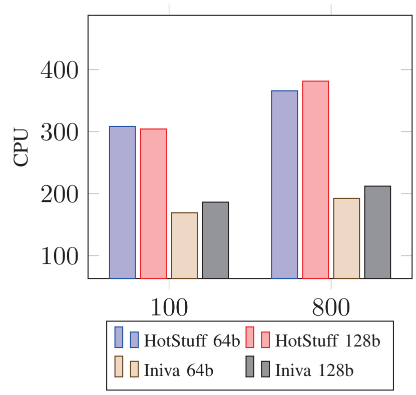
[1]The source code for the experiments and simulations is available at https://github.com/relab/iniva-artifacts.

We also implemented a few variants of Iniva to evaluate our design choices. In most BFT protocols, the leader stops collecting/waiting for votes once it has a quorum. Iniva triggers a 2ND-CHANCE after obtaining a QC to provide a second chance to the processes which their parents intentionally left out. To understand the overhead of this design choice, we implemented a variant that we call Iniva-No2C, where no 2ND-CHANCE messages are sent. Iniva-No2C provides the cost of proposal dissemination and vote aggregation in the tree communication model.

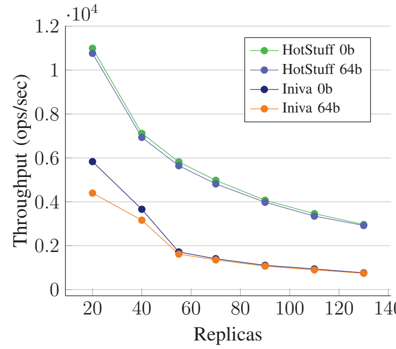
The aggregation timer started on Line 15 of Algorithm 1 determines performance and inclusion of the protocol. If the timer is set too low, the leader may not be able to collect a QC, causing a view failure. If it is set too high, the processes will wait longer for the contribution from faulty processes, resulting in degraded performance. For failure scenarios, we varied the timer to understand its effect on view failures, throughput, latency, and inclusion.



(a) Throughput vs latency for HotStuff, Iniva, and Iniva-No2C.



(b) CPU usage of HotStuff and Iniva.



(c) Scalability evaluation with batch size $B = 100$ and varying configuration sizes and payloads. Configuration sizes are selected to have an almost complete tree of height 3.

Fig. 3: Experimental results with 21 replicas, 4 clients and different payload and batch sizes.

B. Setup

We used our local cluster to evaluate our implementation. The cluster contains 25 machines and each node has 32 GB of RAM and 12 cores of Intel Xenon processors with a maximum frequency of 3.3 GHz. A 10 Gbps TOR switch connects nodes and the latency among the nodes is less than 1 ms. We used round-robin leader rotation policy in the experiments, except for a few experiments where we used the Carousel leader election policy [10]. All experiments run for 150 seconds and metrics are collected every second. The first 5 seconds are used as a warm-up period. All results have less than 1% variance with a 90% confidence interval.

C. Evaluation

We evaluated our implementation in three ways, each with a different objective.

- Base evaluation is performed to evaluate the overhead, throughput, latency, and resource utilization of Iniva and compare it with HotStuff in a fault-free configuration.
- Scaling experiments are conducted to compare Iniva and HotStuff with increasing configuration size.
- Resiliency evaluation is conducted on the different Iniva variants to understand the effect of failures on throughput, latency, and inclusiveness.

1) Base Evaluation: We used 21 machines as processes and 4 machines as clients. For Iniva, these 21 processes are arranged as a complete tree of height 2 with 4 internal and 16 leaf nodes.

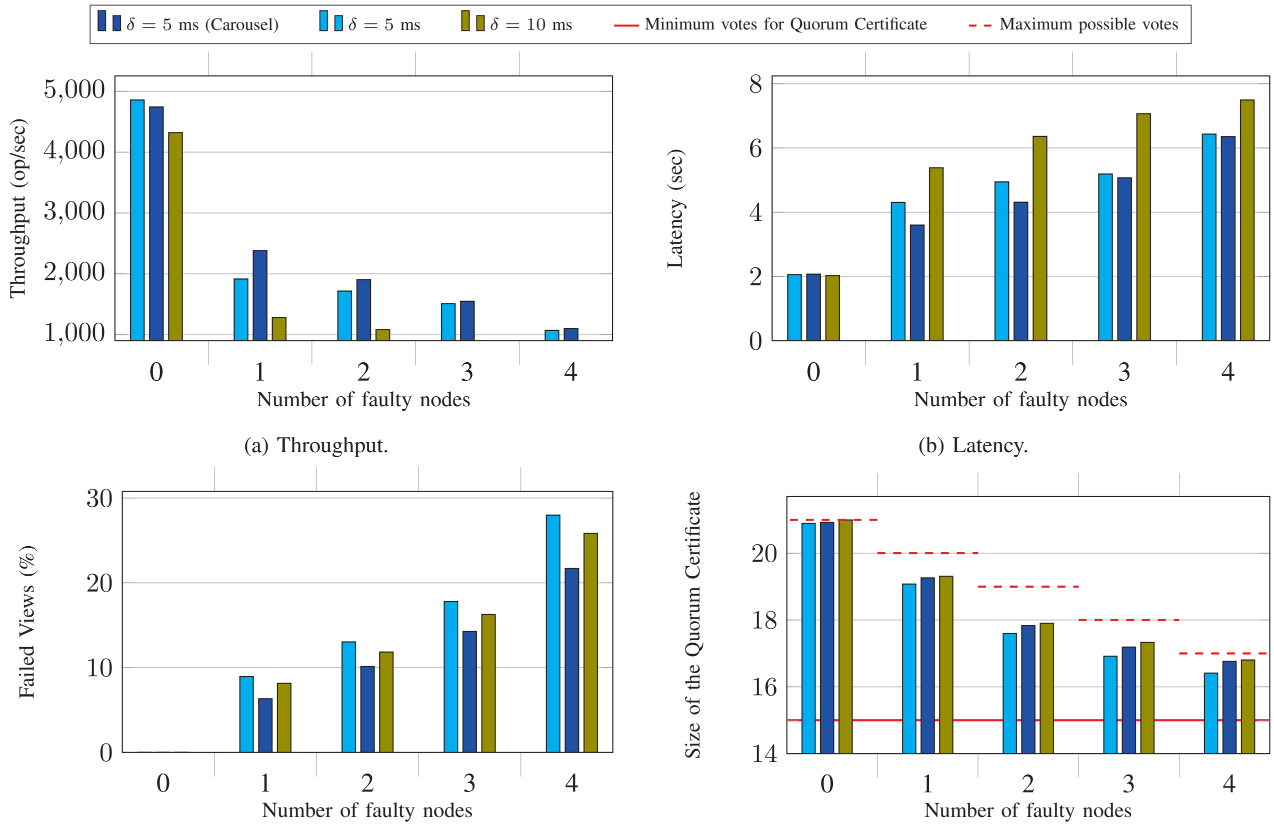
Clients send the request to all processes and expect a quorum of replies before considering the request committed. Requests contain 64 or 128 bytes payload. Batching of requests is enabled at the processes and we used 100 and 800 batch sizes for this evaluation. Clients measure latency and the throughput is measured at the processes. We used BLS12 [11] for signature aggregation.

Figure 3a shows throughput and latency under different client loads. The aggregation timer is adjusted based on the client load on the cluster. We observe that the throughput of Iniva is $\sim 33\%$ lower than HotStuff. The tree-based communication without 2ND-CHANCE (Iniva-No2C) is responsible for about half of the overhead. Although throughput is not the primary objective of Iniva, it can be compensated for with larger batch sizes. Additionally, pipelining of requests in the tree, similar to Kauri [12] could improve throughput. Also, Iniva still has a reasonable throughput compared to most PoW-based schemes such as Bitcoin.

Figure 3b shows the CPU usage for HotStuff and Iniva for two different payload sizes (64 and 128 bytes) and batch sizes ($B = 100$ and $B = 800$). The CPU usage is measured as the percentage of CPU time used by the process. The results show that Iniva uses $\sim 48\%$ less CPU compared to HotStuff. The lower CPU consumption is due to Iniva's tree structure. The tree structure distributes the load and thus reduces CPU usage, but also increases latency and reduces throughput. Doubling the payload from 64 to 128 bytes does not significantly impact CPU usage. When the throughput results are correlated with the CPU usage, we argue that Iniva could outperform HotStuff in a resource-constrained environment.

2) Scaling Evaluation: To evaluate the scalability, we run up to 130 processes, having each physical machine hosting 5 processes. We use batch size 100 and 4 clients. With increased configuration size, the branching factor of the tree is increased to keep the tree's height constant. Figure 3c shows throughput observed for various configurations with and without payload for HotStuff and Iniva. With increased configuration size, throughput decreases gradually.

3) Resiliency Evaluation: We conducted the resiliency evaluation of the Iniva protocol by inducing crash failures in the configuration. As explained earlier, Iniva reconfigures the position of the processes in the tree for every view and faulty processes are randomly placed in the tree. The experiment is done with 21 processes, each running on individual machines with batch size 100 and 4 clients. We set the aggregation timer and second chance timer based on the following heuristic.



(a) Throughput.

(b) Latency.

(c) Percentage of failed views.

(d) Average number of votes included in quorum certificates.

Fig. 4: Experiments with a 21-replica configuration with faulty nodes randomly placed in the tree. We vary the second chance timer (δ) and leader election policy (Round-Robin and Carousel).

Let Δ be the network delay between the processes. The aggregation timer is set to $2\Delta \cdot \text{height}(p)$, where $\text{height}(p)$ is p's height in the tree. The second chance timer is set to $\delta = 2\Delta$. We repeated the experiments with two different δ values, 5ms and 10ms.

Figure 4 shows the effect of failures on the throughput, latency, failed views, and inclusion. With faulty processes in the system, internal processes will wait for votes and the leader will wait for 2ND-CHANCE messages. With increasing failures, latency increases and throughput decreases, as seen in Figures 4a and 4b. The longer second chance timer of 10 ms causes higher latencies and lower throughput.

Figure 4c shows the percentage of failed views. A view may fail either because its leader is faulty, or because no QC could be collected. We also included a variant of Iniva that uses the Carousel leader election to avoid electing faulty leaders. If two of the four internal processes are faulty, no QC can be collected without the 2ND-CHANCE messages. With a higher second chance timer, the number of failed views decreased by 10%.

One of the main objectives of the Iniva mechanism is inclusion. Figure 4d shows the average number of votes included. With 4 failures Iniva includes more than 99% of correct processes. Our baseline, HotStuff, always includes a quorum of 15 votes. We also see that the increased timer has a positive effect on inclusion.

IX. Conclusion

In this paper we proposed Iniva, a vote aggregation protocol to defend against targeted vote omission attacks. Iniva is built upon *Indivisibility*, a feature of some multi-signature aggregation schemes that we defined. Using a tree overlay and fallback paths, Iniva stays inclusive and fault-tolerant. The designed rewarding mechanism motivates processes to participate in the aggregation procedure, and makes Iniva incentive compatible. We conducted several experiments and simulations to analyze Iniva from different perspectives such as security, throughput, latency, recourse efficiency, scalability, and tolerating faults. The results show while Iniva outperforms previous work in terms of preventing vote omission attacks, it has a reasonable performance even in presence of faulty processes in the system.

REFERENCES

[1] V. Buterin, “Ethereum 2.0 spec–Casper and sharding,” *Available [online].[Accessed: 30-10-2018]*, 2018.

[2] Ethereum, “The merge,” Nov 2022. [Online]. Available: https://ethereum.org/en/upgrades/merge/

[3] J. Kwon and E. Buchman, “Cosmos: A network of distributed ledgers,” *URL https://cosmos. network/whitepaper*, 2016.

[4] J. Chen and S. Micali, “Algorand: A secure and efficient distributed ledger,” *Theoretical Computer Science*, vol. 777, pp. 155–183, 2019.

[5] Y. Liu, J. Liu, Z. Zhang, and H. Yu, “A fair selection protocol for committee-based permissionless blockchains,” *Computers & Security*, vol. 91, p. 101718, 2020.

[6] Y. Amoussou-Guenou, A. Del Pozzo, M. Potop-Butucaru, and S. Tucci-Piergiovanni, “On fairness in committee-based blockchains,” *arXiv preprint arXiv:1910.09786*, 2019.

[7] V. Buterin, D. Reijsbergen, S. Leonardos, and G. Piliouras, “Incentives in Ethereum’s hybrid Casper protocol,” *International Journal of Network Management*, vol. 30, no. 5, p. e2098, 2020.

[8] V. Buterin, “Discouragement attacks,” *ETH research*, 2018.

[9] A. Baloochestani, L. Jehl, and H. Meling, “Rebop: Reputation-based incentives in committee-based blockchains,” in *IFIP International Conference on Distributed Applications and Interoperable Systems*. Springer, 2022, pp. 37–54.

[10] S. Cohen, R. Gelashvili, L. K. Kogias, Z. Li, D. Malkhi, A. Sonnino, and A. Spiegelman, “Be aware of your leaders,” in *International Conference on Financial Cryptography and Data Security*. Springer, 2022, pp. 279–295.

[11] P. S. L. M. Barreto, B. Lynn, and M. Scott, “Constructing elliptic curves with prescribed embedding degrees,” Cryptology ePrint Archive, Paper 2002/088, 2002, https://eprint.iacr.org/2002/088. [Online]. Available: https://eprint.iacr.org/2002/088

[12] R. Neiheiser, M. Matos, and L. Rodrigues, “Kauri: Scalable BFT consensus with pipelined tree-based dissemination and aggregation,” in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021, pp. 35–48.

[13] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, “Enhancing Bitcoin security and performance with strong consistency via collective signing,” in *25th usenix security symposium (usenix security 16)*, 2016, pp. 279–296.

[14] O. Bégassat, B. Kolad, N. Gailly, and N. Liochon, “Handel: Practical multi-signature aggregation for large byzantine committees,” *arXiv preprint arXiv:1906.05132*, 2019.

[15] P. Li, G. Wang, X. Chen, F. Long, and W. Xu, “Gosig: a scalable and high-performance byzantine consensus for consortium blockchains,” in *Proceedings of the 11th ACM Symposium on Cloud Computing*, 2020, pp. 223–237.

[16] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, “Hotstuff: BFT consensus with linearity and responsiveness,” in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019, pp. 347–356.

[17] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” Tech. Rep., 2008.

[18] Y. Meng, Z. Cao, and D. Qu, “A committee-based byzantine consensus protocol for blockchain,” in *2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS)*. IEEE, 2018, pp. 1–6.

[19] J. Kwon, “Tendermint: Consensus without mining,” *Draft v. 0.6, fall*, vol. 1, no. 11, 2014.

[20] M. Baudet, A. Ching, A. Chursin, G. Danezis, F. Garillot, Z. Li, D. Malkhi, O. Naor, D. Perelman, and A. Sonnino, “State machine replication in the Libra blockchain,” *The Libra Assn., Tech. Rep*, 2019.

[21] 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*. Springer, 2019, pp. 23–41.

[22] T. Hanke, M. Movahedi, and D. Williams, “Dfinity technology overview series, consensus system,” *arXiv preprint arXiv:1805.04548*, 2018.

[23] M. Castro, B. Liskov *et al.*, “Practical byzantine fault tolerance,” in *OsDI*, vol. 99, no. 1999, 1999, pp. 173–186.

[24] L. Lamport, R. Shostak, and M. Pease, “The byzantine generals problem,” in *Concurrency: the works of Leslie Lamport*, 2019, pp. 203–226.

[25] Y. Amoussou-Guenou, B. Biais, M. Potop-Butucaru, and S. Tucci-Piergiovanni, “Rational behavior in committee-based blockchains,” *Cryptology ePrint Archive*, 2020.

[26] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, “Hotstuff: Bft consensus in the lens of blockchain,” 2019.

[27] V. Buterin and V. Griffith, “Casper the friendly finality gadget,” *arXiv preprint arXiv:1710.09437*, 2017.

[28] I. Abraham, K. Nayak, and N. Shrestha, “Optimal good-case latency for rotating leader synchronous bft,” *Cryptology ePrint Archive*, 2021.

[29] N. Giridharan, F. Suri-Payer, M. Ding, H. Howard, I. Abraham, and N. Crooks, “Beegees: stayin’alive in chained bft,” in *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing*, 2023, pp. 233–243.

[30] C. Dwork, N. Lynch, and L. Stockmeyer, “Consensus in the presence of partial synchrony,” *J. ACM*, vol. 35, no. 2, pp. 288–323, apr 1988. [Online]. Available: https://doi.org/10.1145/42282.42283

[31] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, “Aggregate and verifiably encrypted signatures from bilinear maps,” in *International conference on the theory and applications of cryptographic techniques*. Springer, 2003, pp. 416–432.

[32] D. Boneh, B. Lynn, and H. Shacham, “Short signatures from the Weil pairing,” *Journal of cryptology*, vol. 17, no. 4, pp. 297–319, 2004.

[33] J.-S. Coron and D. Naccache, “Boneh et al.’sk-element aggregate extraction assumption is equivalent to the diffie-hellman assumption,” in *Advances in Cryptology-ASIACRYPT 2003: 9th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, November 30–December 4, 2003. Proceedings 9*. Springer, 2003, pp. 392–397.

[34] S. Micali, M. Rabin, and S. Vadhan, “Verifiable random functions,” in *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*. IEEE, 1999, pp. 120–130.

[35] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and A. Spiegelman, “Solidus: An incentive-compatible cryptocurrency based on permissionless byzantine consensus,” *CoRR, abs/1612.02916*, 2016.

[36] I. Eyal and E. G. Sirer, “Majority is not enough: Bitcoin mining is vulnerable,” *Communications of the ACM*, vol. 61, no. 7, pp. 95–102, 2018.

[37] H. Gogada, H. Meling, L. Jehl, and J. I. Olsen, “An extensible framework for implementing and validating byzantine fault-tolerant protocols,” in *Proceedings of the 5th Workshop on Advanced Tools, Programming Languages, and PLatforms for Implementing and Evaluating Algorithms for Distributed Systems*, ser. ApPLIED 2023. New York, NY, USA: Association for Computing Machinery, 2023.