Harmony Technical Report[*]

Version 2.0

Zeta Avarikioti, Dimitris Karakostas

June 4, 2022

Abstract

Harmony is a Proof-of-Stake sharded blockchain system that aims for high scalability. The system is implemented in practice and has a market cap of about $1 billion at the time of writing. As many of the design decisions of the protocol are described across blog posts and the non-technical whitepaper, the technical knowledge of how the protocol works is scattered. The descriptions in these resources are also high-level and do not give precise algorithms and concrete parametrizations. Additionally, many of the references in blog posts describe Harmony's *vision*, but several of the features are still under implementation. This technical report aims to fill this gap. In the present document, we document the current protocol as it is implemented in the code base, with all its details. We also describe, and explicitly separate, the current state of the code base from any future plans that the team has. Contrary to the code base, which sets the reference implementation, the aim of the current paper is to describe both the *how* and the *why* behind every technical decision. Next, we pinpoint security issues with the current implementation. For each potential security issue, we give a detailed description, and put forth solutions that can ensure better security, towards a provably secure ecosystem.

1 Overview

Harmony is a sharded Proof-of-Stake Layer-1 blockchain. With over $1 billion in market cap at the time of writing, it is a practically deployed system with somewhat wide adoption. Its native token is the *ONE*. This report describes the Harmony protocol in technical detail. The intended audience are scientists in the fields of cryptography, security, and distributed systems, who wish to understand the details and inner workings of the system and the rationale behind its technical design decisions.

[*]This technical report was commissioned and funded by Harmony, and completed by Common Prefix.

The system operates in epochs. For every epoch, each stakeholder may actively participate in the system, by stating the amount they are willing to stake. The 1000 stakeholders with the highest stake (or "bid") participate in the next epoch in two ways: i) they are assigned a slot to "lead", i.e., for which to produce a block; ii) they form a committee which will validate and sign blocks. The committee voting power of each validator in each shard is derived from the staked amount that won its slot. This introduces a new Sybil-resilience stake-based mechanism, termed *"Effective Proof-of-Stake"*.

Thereafter, the slots are assigned to shards effectively determining the validators of each shard. The validators are partially shuffled among shards every new epoch as a new auction and a new slot assignment occur. For this purpose, Harmony employs the *bounded cuckoo's rule* introduced in RapidChain [ZMR18].

The current implementation defines 4 shards in total one of which, called the *"beacon"* shard, handles the validators' auction, records, and computes any information necessary for the protocol's execution. Each client chooses the shard that will process their transaction. The verification of possible cross-shard transactions can be done securely and efficiently only for single-input transactions. For multi input transactions yanking to a single-shard and execution needs to be done [ACDCKK18, ZMR18].

After the slot auction is completed and validators are assigned to shards, they collectively process and validate transactions. For this purpose, a modification of PBFT, called *FBFT*, is introduced, that leverages BLS signature aggregation. Whenever a shard produces a block, the validators send a compressed version of the block, along with its signatures, to the beacon shard. This information is recorded on the beacon shard and used at the end of the epoch to compute the rewards or punishments (i.e., *slashing* or participation exclusion) of each validator. The transaction fees are burned and a fixed reward of new coins is issued per block.

Scope and organization of the report

The purpose of this report is to analyze Harmony's system in detail, and either prove the security of each component or identify potential vulnerabilities and suggest solutions to address them. Towards this end, we dissect each primitive employed by Harmony's sharded consensus.

Harmony's sharded consensus can be compartmentalized in multiple components, each one studied in the next sections. Each section describes: i) the current implementation (and, if applicable, accepted proposals not yet implemented); ii) hazards and attacks against the current implementation; iii) proposed changes and fixes to address the identified issues. In some sections we also discuss already-planned updates, which may mitigate some vulnerabilities on the current implementation.

In particular, the report is organized as follows:

- In Section 2, we present the security model and network assumptions of the system.

- In Section 3, we investigate the novel Sybil-resilience mechanism, namely the Effective Proof-of-Stake.
- In Section 4.1.2, we analyze FBFT, the consensus protocol employed within each shard.
- In Section 5 we focus on sharding and examine all its essential components.
- In Section 6 and Section 7, the reward and punishment mechanisms are investigated, respectively.

Current operation. At the current state of the network, Harmony controls 49% of all voting power.[1] Therefore, the system is almost centralized and various of the hazards and attacks identified below may be not applicable. Nonetheless, the below analysis is informative on the current drawbacks of the system, particularly given the strong commitment to move towards decentralization.

Version 1. This was a preliminary version of the report. Specific components are analyzed in detail, while others are treated in high-level, depending on their "priority". In the next versions of the report, an elaborate analysis will be conducted for each component of Harmony's system and their interoperability will be examined further.

Version 2. We removed the inaccurate "debt" record description. We added a discussion on replay protection for double-signing reports with respect to blacklisting. We clarified assumptions regarding validators' utility with respect to including transactions in blocks. Lastly, we modified the description and proofs of the FBFT protocol for accuracy and completeness.

2 Model

The protocol is operated by n processors, and it proceeds in rounds (asynchronous steps – block generation). $2^{15} = 32{,}768$ consecutive rounds define an *epoch*.

2.1 Network model

We assume a *partially-synchronous network*. Specifically, any message m can be delayed by at most a fixed amount of time, which is unknown to the protocol's designers (and the processors). It has been shown that this is equivalent to a network that alternates between periods of asynchrony and synchrony [DLS88]; in particular, for each execution the network is asynchronous until a Global Stabilization Time (GST), unknown to the processors, after which all messages are delivered within time Δ (known to the processors and protocol designers).

[1]https://talk.harmony.one/t/on-hold-hip-19-100-external-voting-power-and-100-external-slots/4330

2.2 Adversarial model

Honest majority assumption. We assume a byzantine adversary A. At any point in time, A controls t parties. Given the total number n of parties, it holds[2] that: $n \geq 4 \cdot t + 1$.

Furthermore, the adversary is *slowly adaptive*, i.e., at every epoch change, A can flip the corruption state of a constant fraction of nodes[3].

Additionally, we assume a fully adversarial network. In particular, A controls message delivery, such that:

- A controls the order in which messages are delivered to each party at any moment;
- A can delay *any message* under the network's restrictions, i.e., arbitrarily long in the asynchronous phase and at most by Δ in the synchronous phase (after GST).

2.3 Real-world implementation

In practice, the Harmony protocol is executed in rounds, each lasting 2 seconds. Within a single round, a block should be created and collectively signed by an elected committee [Hara].

A validator, i.e., a user that participates in the implemented Harmony protocol, is represented in the system via units of stake. Therefore, following the above model, each party represents a unit of stake, and each validator controls a number of parties proportional to their controlled stake. Thus, the adversary A is assumed to control strictly less than 33% of the total stake in the system.

3 EPoS: The Sybil-Resilience Mechanism

3.1 Current Implementation

3.1.1 PoS participation

To guarantee security in a permissionless setting Harmony employs a Proof-of-Stake Sybil-resistance mechanism, called Effective PoS (EPoS).

A validator V is a party that participates in consensus by performing transaction and block verification [Harf]. Each validator registers on the ledger via a *CreateValidator* transaction, which defines:

1. *amount*: the amount of ONE tokens V stakes, i.e., V owns and locks,
2. *rate*: the commission fee (see below Section 3.1.3),
3. *bls-pubkeys*: the BLS public key(s), with which V participates and signs messages.

[2] Looking forward, this assumption, together with the *sharding mechanism* (Sec. 5), ensure that *within each shard* the adversary controls less than 1/3 of the parties.

[3] An adversary that can dynamically *uncorrupt* parties is also commonly known as *mobile*

Each party, who acts as a validator, may register multiple BLS keys. Each such key makes a single bid. Specifically, consider a validator V, who controls (via self-staking and delegation) s units of stake and has registered b BLS keys; during the slot auction, the bidding price of each key is automatically set at $\frac{s}{b}$. Per registration, each validator can define a number of BLS keys at most equal to $\frac{1}{3}$ of the total available seats [Harf] of the epoch's validating committee.

At the beginning of each epoch, e slots are auctioned for the prospective validators of the shards. The e highest bidders are each awarded one slot. Therefore, the e highest-bidding BLS keys form the epoch's committee, each key being the leader of a single slot. The shard on which each key is assigned is chosen via a modulo operation on the key's value [Hare].

Validators are also allowed to run clones of their node instances, which control the same BLS key(s), for redundancy and robustness purposes. However, an implicit assumption is that clones should coordinate, in order to maintain the same state and avoid producing or signing conflicting blocks.

3.1.2 Effective stake

Both the block reward and the voting power of each validator is proportional to their effective stake ϕ_i. Specifically, after the bidding process ends, the effective stake ϕ_i of each elected validator (i.e., BLS key) is determined as follows [Lanb]:

$$\phi_i = \max(\min((1 + c) \cdot m_s, s_i), (1 - c) \cdot m_s)$$

where s_i is the validator's bid for the slot, m_s is the median successful bid for all of the epoch's slots, and c is a protocol parameter. In the current implementation $c = 0.15$ and $e = 1000$.[4] Note that only the bids of the elected validators are used to calculate m_s.

The goal of this validator election process is to guarantee a fair assignment of slots, i.e., each stakeholder should gain a slot with probability proportional to their stake. This is necessary to guarantee the security bounds when transitioning from a permissionless to a permissioned system (which is the setting under which the validation committee of each epoch operates).

Example attack. Consider the case in which the bid of P_i is $s_i = i, \forall i \in [1000]$. The total real stake is $\sum_{i=1}^{1000} = 500500$. Then $m_s = 500$, so $(1-c)\cdot m_s = 425$, $(1+c)\cdot m_s = 575$, therefore $\forall i \in [425], \phi_i = 425$, $\forall i \in \{426, \ldots, 575\}, \phi_i = i$, $\forall i \in \{575, \ldots, 1000\}, \phi_i = 575$. The total effective stake is $425^2 + \sum_{i=426}^{575} i + 575(1000 - 575) = 500075$. Assume now that the adversary controls just below 1/3 of the real stake, i.e. 166833 coins in total. In her attempt to game the system, she has submitted the 578 smaller bids. Her effective stake is 257425, i.e. 51.5% of the total effective stake. We conclude that an adversary can control over a 1/2 majority of the *effective* stake, even though she controls below 1/3 of the *actual* stake.

[4] Out of the 1000 slots, the 100 are internal controlled by Harmony, while the rest 900 are available for auction to external stakeholders.

3.1.3 PoS delegation

A user that wants to participate in staking, but does not want to become a validator, can delegate its stake [Lana]. Delegation is activated via special transactions published on the ledger. A delegated token is locked for the duration of the delegation and shortly thereafter (see below). For every delegated token, the user receives a fixed reward (see Section 6); the rewards are collected in a separate account and are immediately available to the user for withdrawal.

Undelegation and redelegation. When a user wishes to undelegate its stake, they issue a special transaction. Following, the tokens remain locked for a period of 7 epochs. If the user wishes to redelegate their stake instead, they can do so almost immediately, i.e., from the beginning of the first epoch after the "undelegation" transaction is issued [Lanc], and without having to wait for 7 epochs. One exception is that, if the validator to whom the user delegates has never been elected as part of a committee, the token is undelegated immediately.

3.2 Current Hazards and Attacks

3.2.1 Adversarial takeover under favourable stake distribution

Given there are no assumptions on the distribution of the stake, suppose all honest stakeholders hold (the minimum of) one coin, which they operate themselves, i.e., do not delegate. However, suppose that the adversary A holds 2000 coins. A can make 1000 bids of 2 coins each. In turn, each honest stakeholder can bid at most 1 coin. Note that, as long as the number of honest stakeholders is large enough (greater than 6000), A holds less than (the threshold of) $\frac{1}{4}$ of the total stake. However, the adversary clearly gains control of the system, as her bids are the top 1000, the amount needed to control all slots. Naturally, this is a non-realistic cornercase which we do not expected to ever manifest in practice; nonetheless, it is worth noting while evaluating the theoretical limits of the mechanism.

3.2.2 Incentive incompatible auction for validator election

Each validator "pays" to the mechanism its bid, in other words, the validator locks the stake he bid to the mechanism. The highest bidders win the first 1000 slots[5]. This is akin to a first-price auction, with the exception that bids are public since the moment they are posted (i.e., they are not submitted hidden and revealed altogether, as in classic first-price auctions).

A well-known result in game theory shows that first-price auctions are not incentive-compatible [Rou16]. In particular, rational bidders are inclined to underbid and not submit their truthful valuation for the auctioned item. In the current implementation, each slot's leader is elected via a first-price auction, so

[5] 1000 slots in total when the protocol is fully decentralized

the dominant strategy for all stakeholders is to underbid. In the context of our setting, "underbidding" means that stakeholders lock less stake.

Underbidding is important, because the system's security relies on the following assumption: "The attacker does not control more than $\frac{1}{3}$ of the total *bid* stake.[6]" If (rational[7]) parties underbid, the total amount of (rational) stake that is bid is smaller, thus the amount of stake the adversary needs to control is reduced.

In generic auction mechanisms we examine the following properties:

- *Individual Rationality (IR):* No player should lose from participating in the auction.
- *Budget Balance (BB):* The auctioneer should not lose/gain any money.
- *Truthfulness or Incentive-compatibility (DSIC[8]):* Reporting their true value is the dominant strategy for all players, i.e., a player should not be able to gain by spying over other players and trying to find an 'optimal' declaration which is different from its true value, regardless of how the other players play.
- *Economic efficiency (EE):* The total social welfare (the sum of the values of all players) should be the best possible. In particular, this means that, after all trading has completed, the items should be in the hands of those that value them the most.

It is a well-known result that all these properties cannot be achieved simultaneously [MS83].

Currently, it is unclear how these properties apply in the context of blockchain systems. The standard auction setting is a one-off execution and each party's valuation is typically private. In our case, the election mechanism consists of multiple repeated slot auctions. On each auction the same parties will typically take part with roughly the same preferences. Therefore, after a sufficient amount of time, it is expected that all parties will learn the valuations of all other parties. Nevertheless, even considering public auctions is not straightforward, as an interpretation of game-theoretic properties to consensus security properties is necessary. As such, the above properties are not directly applicable; further research is needed to identify the desired properties for this setting and evaluate whether the implemented system satisfies them.

[6]The amount of stake that is bid is typically less than the total stake, as some coins are actually used in transactions. However, the validators' election and voting power relies on the stake that they lock (or the locked stake they are delegated). Therefore, a potential attacker, who aims at disrupting consensus, needs to control more than $\frac{1}{3}$ of the stake that "participates" in consensus, i.e., the locked stake.

[7]Rational parties intentionally deviate from the protocol specification only if they can maximize their utility, i.e., profit; otherwise, they simply act honestly.

[8]DSIC stands for Dominant Strategy Incentive Compatibility, the strongest form of IC.

3.2.3 Validator Sybil Attack

As there is no centralized identity management, a party may create as many identities as they want. For each identity, they might also register multiple BLS keys, to act as validators.

By creating multiple, seemingly independent identities, an adversary A might lure honest stakeholders to delegate their stake to A's BLS keys. Since the bid is computed over the key's controlling stake, i.e., self plus delegated stake, an attacker may attract a large amount of delegated stake, and thus control a large part of the slots, by owning only a small amount on their own.

In existing literature [BKKS20], this has been bypassed by assigning a score to each validator, based on their self stake. In particular, validators that self-stake a large amount are more preferable compared to "smaller" validators. This mechanism prevents an attacker from splitting their stake to create multiple validators. However, it should be noted that it also promotes "richer" validators.

3.2.4 Long-range attacks

There are two kind of long-range attacks in PoS protocols. These attacks seem to apply also in Harmony's protocol, similarly to other PoS systems.

Dynamic availability: Let us assume the "active" stake is variable, meaning that at one epoch e_0 the total active stake is 10% and at a much later epoch e_1 it reaches 100%. Now assume that the adversary controls 25% of the (total) stake in both cases. This means that at epoch e_1 the adversary has 2.5 times the entire stake of epoch e_0. Consequently, the adversary can, in theory, rewrite the entire history from epoch e_0 to epoch e_1 with minimum cost.

Bribing attack: The adversary can at a later time buy the keys that correspond to honest participants in an early epoch, acquiring more voting power than the honest participants for that epoch. As a result, the adversary can rewrite the history from that point on.

Both these attacks are feasible due to the costless simulation of PoS protocols and the dynamic nature of the network (new nodes join and leave). To this date, we are not aware of a solution that does not introduce additional restrictions (e.g., erasure of old keys), trust assumptions (e.g., the entire honest stake is always "active"), or computational tools (e.g., Proof-of-Work or VDFs).

Note: Harmony plans to erase their own keys that controlled the majority of the voting power during the times of the network launch.

3.3 Planned Updates

Order of participation. The "resharding" planned release will change the way validators are elected to participate. Specifically, on each epoch a random seed will be produced by the elected validators. For each epoch, the order of how validators participate will be based on a shuffling of all committee members, seeded by the previous epoch's randomness (see Section 5.1.5).

3.4 Proposed Changes

To address the vulnerabilities against rational participants, a novel analysis of desired properties and how to achieve them is necessary. This analysis is currently undergoing and will be included in a followup report.

4 Permissionless Consensus

4.1 Current Implementation

4.1.1 Distributed Randomness Generation

Currently, the Distributed Randomness Generation (DRG) is implemented by a Verifiable Random Function (VRF). The specification of this protocol is currently under review.

4.1.2 Consensus (intra-shard)

The consensus protocol is termed Fast Byzantine Fault Tolerance (FBFT) and it is a modification of PBFT [CL02] where the leader employs BLS signatures [Bon05] to aggregate the signatures it collected from the validators in every step. We provide a description below, while for more details see [Hard, Harc].

FBFT normal operation: The FBFT protocol, under normal conditions, operates as follows.

***Announce* phase.**

1. The leader constructs the new block and broadcasts the block hash to all validators. The leader also compresses the block with erasure coding and then broadcasts it to all validators.

***Prepare* phase.**

2. The validators, upon receipt of the block header, check that the block is at the correct round, sign the block hash with a BLS signature, and send the signature back to the leader.
3. The leader waits for valid signatures with more than 2/3 voting power from validators (including the leader itself) and aggregates them into a BLS aggregate signature. Then the leader broadcasts the new block and the aggregated signature along with a bitmap indicating which validators have signed. Concurrently the validators have received the block content and start the verification.

***Commit* phase.**

4. The validators, upon receipt of a prepared message, check that the aggregate signature has at least 2/3 of total voting power, and wait for the validation of the block content (broadcasted by the leader in Step 1). Once they know that the block is available and valid they sign the received block from Step 3, and send it back to the leader (e.g. signature on $\langle \mathsf{blockNum}, \mathsf{blockHash} \rangle$).

5. The leader waits for valid signatures with more than 2/3 voting power (these can be different signers from Step 3), aggregates them together into a BLS aggregate signature, and creates a bitmap for all the signers. Finally, the leader commits the new block with the aggregate signatures and bitmaps attached, and broadcasts the aggregate signature and bitmap for all validators to confidently commit to the block.

Remarks:

- In step 3 commit phase, the validator sends a commit message with a signature on the blockNumber and the blockHash. This way the node can quickly determine whether it is out of sync without being tricked by a malicious leader.

- The leaders that perform the consensus are pre-determined by the output of the Sybil-resistant function; the DRG determines the order of leaders in each shard. If a view change is necessary because a leader timed-out, the next leader in the pre-determined order proceeds.

FBFT view change: The view change protocol does not increase timers like PBFT does. Instead, it counts epochs as the difference between the local time of the node and the time the last block was committed.

The view-change protocol operates as follows:

1. When the consensus timer timeouts, a node starts the view change by sending a view change message. The message always has a signature on the viewID (m3) and either a signature on NIL (m2) signaling that there is no available prepared block (if normal operation failed before the completion of the "prepare" phase) or a signature on the hash of the prepared block (m1) (if normal operation failed after the completion of the "prepare" phase). The validator also forwards the prepared block and the prepared signature.

2. When the new leader receives enough matching ($\geq 2f+1$) view change messages, it aggregates the signatures of viewID and selects the newest prepared message from the view change messages. Then, the leader broadcasts the new view message including the aggregated signatures as well as the selected prepared message. Thereafter, the new leader switches to normal consensus mode. A validator switches to normal consensus mode

when it receives a new view message from the new leader; at the same time, it stops the view change timer and start the consensus timer. If the validator does not receive a new view message before the view change timeout, it will increase the viewID by one and start another view change.

3. The leader cannot cheat since the validators send either an m2 or an m1 message, hence if a block is committed then $m3count > m2count$ and the leader has to reveal a prepared block. If there are multiple m1 blocks then some slow nodes might accept an older prepared block but the commitment on normal mode will fail.

Remarks:

- The second step requires each validator to send a signature on viewID. The purpose is to reduce the size of the new view message from $O(n)$ to $O(1)$ when the previous leader is malicious.
- When a leader is out-of-sync due to network problems, it must sync-in again to be able to participate in the consensus. For this reason, the block headers include the leader and viewID. Then, if a view change happens during the syncing of a node, the information from the latest block is outdated. In this case, a new node can still update the leader and viewID information when it receives a committed message.

Next, we show that FBFT is *safe* in asynchrony and *live* under bounded clock drift. To do so, we first define Safety and Liveness in blockchains following (crudely) the definitions of [GKL15]. Blockchain protocols must generally achieve:

- **Safety:** If a transaction is included in a block of an honest party, then all honest parties will include this transaction in the same block.
- **Liveness:** If a transaction is provided to honest parties for enough consecutive rounds, then the transaction will be eventually included in a block[9].

Theorem 1 *FBFT is safe in asynchrony.*

Sketch proof. We will show that the FBFT protocol is safe in asynchrony. First, we show that the protocol is safe under normal operation and then we prove that no two contradicting blocks can be committed.

Normal operation: The main difference of FBFT and PFBT is that at step 2 (of the normal operation) the validators send their signature on the hash of the block, instead of the block itself. As a result, the hash may be prepared although the block is either never received (data availability) or it is invalid (data integrity). However, during commit the nodes only provide signatures after reception and validation of the block. As a result the normal operation of FBFT and PBFT [CL02] are equivalent.

[9]For liveness guarantees, we assume the block size is infinite.

[Towards contradiction.] Suppose there are two parties P_1, P_2 that commit two contradicting blocks b_1 and b_2 respectively (without loss of generality, a block contains one transaction), so safety is violated. Now there are three cases that this can occur:

1. Neither party did the view-change operation. This is a contradiction, since the normal operation is safe as explained previously.

2. Both parties did the same number of view-change operations before resuming normal operation and committing to their blocks. In this case, both parties agree on who was the leader when they switched back to normal operation, therefore the argument above for normal operation applies, leading again to a contradiction.

3. The two parties resumed normal operation at slots t_1 and t_2 respectively with $t_1 \neq t_2$, i.e. after a different number of view-change operations. Then P_1 received an aggregate signature of at least $2f + 1$ parties for b_1 during the "commit" phase of normal operation at t_1, and so did P_2 for b_2 at t_2. Let s be the number of malicious parties and h the number of honest parties. As per assumption, it holds that $s \leq f$ and $s + h = 3f + 1$. For $i \in \{1, 2\}$, let $0 \leq s_i \leq s$ the number of malicious parties that signed b_i. Then for $i \in \{1, 2\}$, $2f + 1 - s_i$ honest parties signed b_i. For the two sets to be disjoint, it must be $2f+1-s_1+2f+1-s_2 \leq h \Rightarrow 4f+2-s_1-s_2 \leq 3f + 1 - s \Rightarrow f + 1 \leq s_1 + s_2 - s$. But summing $0 \leq s_1 \leq s$, $0 \leq s_2 \leq s$ and $s < f + 1$ gives $s_1 + s_2 - s < f + 1$, thus the two sets are not disjoint. Therefore there is at least one honest party that signed both b_1 and b_2 for the same slot during normal operation, which contradicts the protocol of honest parties. □

Theorem 2 *FBFT is live in synchrony, i.e., assuming a pacemaker.*

Sketch proof. Under the assumption of a correct pacemaker, the validators are synchronized, hence liveness holds similarly to PBFT [CL02]. □

4.2 Hazards and Attacks

4.2.1 FBFT security properties

The timers do not increase in FBFT. This a) does not protect from clock skew; if local timers drift apart, enough from $2f + 1$ the validators will lose liveness as they will never converge in view, and b) does not adjust epoch length; as a result if the network's latency (Δ after GST) is higher, then again the system loses liveness. This is of practical concern since the theoretical definition of weak synchrony assumes knowledge of Δ at the beginning of the protocol.

Note: Since this is only a liveness problem that is unlikely to occur, Harmony does not consider addressing clock skew a priority.

4.2.2 Complexity of FBFT

The communication complexity of FBFT is $O(n^2)$ with a good leader, where n is the number of participants. This is because the leader broadcasts to all validators (n) the bitmap which asymptotically has $O(n)$ complexity. So the improvement in the communication from the signature aggregation is practical but not theoretical. This may not be a concern for the implementation. The worse-case runtime is O(f) for f consecutive view-changes, assuming the clocks of all honest parties are synchronized. This results in worst case communication complexity of $O(n^3)$.

4.3 Proposed Changes

4.3.1 FBFT liveness in asynchrony

The view change needs to wait for slow parties to catch-up and send consistent viewID messages. Since there is no catch-up mechanism for updating views and assuming f corrupted parties, a single party with skewed clock is enough to break the liveness by running ahead and switching views before the leader has enough messages to start the view. In order to solve this, the nodes need to synchronize their clocks. During loss of liveness this can also happen manually using an external communication medium (e.g., landlines, discord, twitter).

5 Sharding

In this section, we discuss all the components of Harmony's sharding system. We identify some vulnerabilities either from a theoretical or practical perspective. We then discuss potential solutions to the vulnerabilities that are deemed more important. However, our analysis is currently on a preliminary stage and thus many details and specific parameters are omitted. This document will be updated after a thorough analysis of all the sharding components and their interoperability is conducted.

5.1 Current Implementation

The current implementation has 4 shards each operated by 250 validators. The system progresses in epochs, a predetermined time interval during which the system operates in a permissioned setting (fixed set of validators). The epoch size is polynomial in the total number of validators across all shards; in the implementation it consists of 32,768 blocks (approximately 18.2 hours in normal operation).

First, the EPoS process ensures the fair division of stakeholders in slots (Section 3.1.2). Hence, from now on the system can be considered permissioned operated by a specific set of validators (1000 validators in total – currently 100 validators are controlled by Harmony and they hold in total 49% of the stake). This process takes place in the beacon chain, which is a shard acting as an identity chain, i.e., determines the validators for the next epoch, and is also responsible for generating the randomness for the next epoch (Section 4.1.1).

Then, the slots/validators are assigned into shards, currently leveraging the BLS keys (Section 5.1.1). Each shard operates independently, meaning that the validators of each shard maintain the accounts only for this shard and verify transactions by executing the intra-shard consensus (Section 4.1.2). The state is not partitioned into the shards but instead each user specifies the shards of the payer and the payee of a transaction (Section 5.1.2). If a transaction spans across multiple shards, i.e., it is a cross-shard transaction, the shard-driven cross-shard mechanism guarantees the atomicity of the transaction; that is, either the transaction goes through in all shards or none of the shards (Section 5.1.4).

Whenever a block is committed in one of the shards, the corresponding block header is sent to the validators of the beacon chain, who verify and store the header on the beacon chain along with the signatures of the block validators (Section 5.1.3).

At the end of each epoch, the validators are shuffled among shards to guarantee security against a *slowly adaptive* adversary. The exact reconfiguration process can be found in Section 5.1.5.

Note: Currently Harmony controls 49% of the voting power in each *shard to ensure that the protocol is secure against potential vulnerabilities. Nevertheless, we treat this exposition and analysis as if the system operates in a decentralized manner, as is planned.*

5.1.1 Division to shards

When assigning validators to shards, the registered BLS keys are used (cf. Section 3.1.1). In particular, each elected BLS key is assigned to a shard via a simple *modulo* operation on the number of shards [Hare]. In the current implementation, the number of shards is 4.

5.1.2 State partition

In contrast to most sharding protocols, Harmony's protocol does not specify how the accounts are partitioned into shards. Instead the user specifies the shards corresponding to the payer and the payee. In particular, the user submitting a signed transaction, specifies a field named `shard_id` which indicates the shard this transaction belongs to. For cross-shard transactions, another field named `to_shard_id` is specified that indicates the shard of the *payee*, while `shard_id` indicates the shard of the *payer*.

Incentives for transaction-balanced shards. Harmony allows its clients to decide which shard they want to open an account in and therefore operate. This strategy does not guarantee balanced shards in terms of the number of transactions to be validated by protocol design. Nevertheless, it is assumed that the clients are rational and will choose the shard that provides them lower fees and latency. As overloaded shards are expected to be more expensive and have higher latency for transaction confirmation, a new client is expected to choose the shard with the least workload.

5.1.3 The beacon chain

The beacon shard performs transaction validation and moreover is responsible for the following operations:

1. **Randomness:** The validators of the beacon chain execute the DRG protocol to generate the randomness of the next epoch.

2. **Epoch validators:** The validators of the beacon chain accept stakes as bids from the stakeholders that want to participate in the next epoch. The EPoS takes place in the beacon chain[10].

3. **Crosslinks:** A *crosslink* is a data structure that contains data for block signatures and the block identifier data such as block hash, block number, view id and epoch etc. When a new block is confirmed in a shard, the corresponding crosslink is created and sent to the beacon chain (via Kademlia-based inter-shard communication). Upon the receipt of the crosslink, the beacon chain verifies its signature and checks that it is from the canonical chain of the shard. Specifically, the beacon chain verifies: (a) the hash of its previous block, which should already be committed to the beacon chain, and (b) the signatures of the validators corresponds to the ones assigned in this shard. Successfully verified crosslinks are added in the new block of the beacon chain to permanently endorse the block of the shard as canonical. Shard blocks without corresponding crosslinks endorsed in the beacon chain will not be recognized by the network and will be deemed as invalid blocks.

   Crosslinks are also used to record and tally the signing activity of the shard chain validators. Since epoch transition and EPoS election happens only on the beacon chain, the validators' signing activity from each shard are sent to the beacon chain via the crosslink to later determine the block reward and uptime (i.e., the fraction of this epoch's blocks the validator has signed) which affect the validator's election status.

5.1.4 Cross-shard mechanism

Cross-shard mechanisms are critical for consistency as they guarantee the atomicity of cross-shard transactions. In other words, the cross-shard protocol makes sure that a transaction is either committed or rejected in all shards.

[10]The tokens staked in the beacon chain (delegated or not) cannot be used for transactions. When 7 epochs pass, these tokens can, however, be transferred into an account and thereafter be used for transacting.

Harmony adopts an account-based transaction and supports only single-input transactions[11]. In this case, the cross-shard mechanism is simple: the input shard (shard of the payer) verifies the transaction and produces a receipt with at least 2/3 of the validator's signatures. The receipt is sent to the output shard(s), i.e., the shard of the payee, that simply verifies that enough correct signatures are present and consequently validates the transaction in the output shard. Note that the validators of each shard are public knowledge as the assignment into shards is recorded on the beacon chain.

5.1.5 Reconfiguration (new epoch)

The reconfiguration process is critical for security to withstand slowly adaptive adversaries. At this stage, the adversary can leave and rejoin the network as well as change the corrupted validators. To defend against the adversarial adaptivity, Harmony adopts the *bounded Cuckoo's rule* from Rapidchain [ZMR18], a modification of the Cuckoo's rule introduced in [AS09].

Cuckoo's Rule [AS09]. To map validators to shards, validators are first mapped to a random position in [0, 1) using a hash function. Then, the range [0, 1) is partitioned into k regions of size k/n; a shard is defined as the group of validators that are assigned to $O(\log n)$ continuous regions, for some constant k. When a validator wants to join the network, it is placed at a random position $x \in [0, 1)$, while all validators in a constant-sized interval surrounding x are moved (or cuckoo'ed) to new random positions in (0, 1]. It has been proven that if a constant fraction of parties $\epsilon < 1/2 - 1/k$ carry out a join/leave operation, all regions of size $O(\log n)/n$ have $O(\log n)$ validators of which less than 1/3 are faulty, with high probability, for any polynomial number of rounds.

Bounded Cuckoo's Rule [ZMR18]. A validator is called *new* while it is in the shard where it was assigned when it joined the system. At any time after that and as long as it does not leave the system, we call it an *old* validator. We define the *age of a k-region* as the amount of time that has passed after a new validator is placed in that k-region. We define the *age of a shard* as the sum of the ages of its k-regions.

Let a validator be *active* during an epoch if it has signed at least a fraction p (e.g. $p = 67\%$) of the epoch's blocks. Also let $m = O(\log n)$ be the number of shards. At the start of each epoch, the beacon chain first populates the set of active validators that remain in the protocol for the new epoch. Then, the beacon chain defines the set of the largest $m/2$ shards (who have more active members) as the active shard set, denoted by A, and the remaining $m/2$ shards with smaller sizes as the inactive shard set, denoted by I. Active shards accept new validators that have joined the network in the previous epoch, as new members of the shard. Inactive shards, on the other hand, only accept

[11]Harmony does not support cross-shard smart contracts only cross-shard transfers [ACDCKK18]

the members who were part of the network before, to join them. For each new validator, the beacon chain chooses a random shard C from the set A and adds the new validator to C. Next, the beacon chain evicts (cuckoos) a *constant number* of members from every shard (including C) and assigns them to other shards chosen uniformly at random from I.

It has been proven that this process ensures a bounded deviation in the shard sizes, and that each shard has an honest majority [ZMR18]. However, the adversarial model is weaker than the typical one – the adversary can only change a *constant* number of corruptions, and only a *constant* number of validators can leave/join the network.

Determining trade-off security and number of leave/join operations: Rapidchain does not provide a proof. Instead it is stated that the following hold with high probability:

- The maximum number of nodes in each committee is $c \log n + c/2(1 + \delta)(3 - \frac{t}{k} + \frac{t}{n-t}k) \log n$, where n is the total number of validators, t the corrupt nodes, k is a constant such that $\frac{t}{n-t} < 1 - 1/k$ (for Harmony's protocol that is $k > 2$), δ a small constant for Chernoff bounds to apply, and c a security parameter.

- Any committee has $(1-t/n)(1-\delta)c \log(nk/2)$ honest and $\frac{t}{n-t}(1+\delta)c \log(nk/2)$ corrupt nodes with high probability.

We observe c is parameter inherited by the original paper [AS09] and we cannot estimate by these bounds what is the correlation of the constant number of join/leave nodes and security guarantees. For this reason, we look at the original notation and bounds [AS09].

- The bounds hold for any region of size $c \log n$, hence c is the constant multiplier that determines the adequate shard size. δ (Chernoff bound parameter) can be made arbitrarily low depending on c.

- The high probability bounds are stated with respect to k.

- *Cuckoo's rule achieves honest and balanced shards against adaptive join/leave attacks of a* constant fraction *of adversarial peers.*

- It is shown that for any logarithmic (with respect to n) interval of time T, the honest majority in a shard is maintained. Lemma 2.8 states that at any time this bound holds (similar to the one stated above from Rapidchain).

- The security holds for polynomially many (with respect to n) rounds (join/leaves). This is translated in Rapichain into polynomially many epochs of constant changes, but that is not necessary for security – polynomially many changes in polynomial number of epochs still give a polynomial number of changes. We also highlight that the security analysis is done under the worst possible adversarial strategy, i.e., the adversary attacks one shard specifically and only leaves and rejoins with its nodes outside of that shard.

Remark: We conclude this strategy is secure enough for practical purposes as long as the security parameters (c, k) are chosen wisely. Nevertheless, the system should occasionally reboot, meaning that once in while (e.g., once per year) a total reassignment of nodes should occur, to maintain long-term security.

Harmony's reconfiguration. At the beginning of each epoch, all slots are eligible for auction. All stakeholders are eligible validators apart from the ones that have a low uptime in the last epoch, i.e., the validators that signed less than a p fraction of the blocks in their shard in the previous epoch (e.g., $p = 67\%$). Note that the validators with low uptime will be excluded at least for one epoch, but, if during the next epoch they ask to be considered active in the epoch after next, their status will be updated from *inactive* to *active.* The auction proceeds as described in Section 3.1.2.

When the new set of validators is determined, the protocol divides them into two categories: *old* validators and *new* validators. The old validators are the ones that won the same slot as in the previous epoch. The new validators are the ones that just won a slot, and that slot belonged to another validator in the previous epoch. The old validators keep their old shard assignment temporarily, while the new ones are assigned one by one to new shards according to the bounded cuckoo's rule. As a result, some of the old validators will also be moved to another shard. This process takes place after the auction to determine the validators that will operate each shard in the next epoch.

5.2 Current Hazards and Attacks

5.2.1 Grinding attacks on BLS keys – single shard take-over

The modulo operation of assigning BLS keys to shards is susceptible to grinding attacks. In particular, each validator can choose the shard on which to participate. Therefore, an adversary can split its stake across multiple BLS keys, which are pre-computed such that all are assigned to the same shard. As a result, an adversary can take over a single shard with a bit more that $\frac{1}{8}$ of the total stake, given 4 shards (as in the current implementation).

5.2.2 Violating the honest majority in a single-shard

The current implementation assumes the number of corrupted nodes in the system t and in each shard f is the same and equal to $1/3$. This holds in expectation but not with high probability, therefore there is a non-negligible probability that one shard has more than $1/3$ corrupted nodes which in turn lead to a single-shard takeover.

5.2.3 Violating the constant number of join/leave/move assumption

All slots are available for auction in each epoch-change. Thus, the constant number of leave/join/move validators may be violated by the protocol design. In this case, there are no guarantees that the shards have an honest supermajority (2/3) or that they are balanced [ZMR18]. Therefore, after a number of epochs we may have an adversarial shard takeover.

Note: The aforementioned violation is partially true. The auction, as is, does violate the Rapidchain's assumption, but this assumption does not seem necessary for security [AS09] as long as the honest nodes are assigned to the same slot every time.

5.2.4 Attacking uptime to gain in EPoS

At the end of each epoch, only the nodes that reached the desired uptime are eligible to be validators for the next epoch. So if a validator has signed less than a p fraction of blocks then its slot will be auctioned again, and this stakeholder will not be an eligible validator at least for the next auction.

Complete adversarial takeover for $p \geq 50\%$. We will show that, if the desired uptime $p > \frac{f+1}{2f+1}$, then the adversary can control all the validators in the next epoch. Suppose B denotes the number of blocks generated in one epoch in one shard, f the number of validators controlled by the adversary in the shard and $n = 3f + 1$ the total number of validators in the shard.

In an epoch, $\frac{f}{n} \cdot B$ blocks are produced by a Byzantine leader, while the rest $\frac{n-f}{n} \cdot B$ have an honest leader. When the adversary is a leader, she excludes f signatures. On the other hand, when an honest validator is a leader the adversary sends her f signatures and delays the message delivery of f parties (more than 3 seconds; 2s is the block generation time and 1s the grace period for gathering more signatures). Therefore, the honest leader will include the f signatures from the adversary and $f + 1$ from honest validators. Thus, each block generated will have f signatures from the adversary and $f + 1$ from the honest. Moreover, the adversary is free to choose which exact signatures will be included every time (either because she is the leader of because she can delay any message delivery she chooses to). Therefore, the total number of signatures belonging to honest validators in one epoch is $(f + 1) \cdot B$.

Now, due to the uptime exclusion, each validator needs at least $p \cdot B$ signatures to be considered active. The adversary will therefore include first 1 signature from each party, then 2 from each validator, 3 and so on, until all of them have 1 less than needed. Then, the adversary will include the signature of one party in as many blocks as possible and then proceed to the signature of a second validator. With this strategy the adversary may maximize the number of honest validators that will be excluded in the next auction.

Intuitively, the adversary alternates between two distinct sets of size f for each block, hence at the end of the epoch each set will have been part in only half the blocks. If the uptime exclusion target is more than 50% then all honest validators will be excluded and the adversary will monopolize the next epoch (and violate the reconfiguration assumption). Formally, $p{\cdot}B{\cdot}(2f{+}1) \leq (f{+}1){\cdot}B$, hence $p \leq \frac{f+1}{2f+1}$.

There is no $p > 0$ such that reconfiguration is secure when $n = 3f + 1$. In the asynchronous phase, the adversary controls which signatures are included in each block both when she is leader (by choosing) and when the leader is honest (by delaying the message delivery of specific nodes). Since a block needs $2f + 1$ signatures to be included, the adversary can target specifically f validators and exclude them every time. These validators will have uptime zero, and thus be excluded in the next epoch. Therefore, the adversary will be competing only against the remaining $f + 1$ honest stakeholders for the validators slots. As a result, the adversary can violate the security bounds and win almost 50% of the slots for the next epoch.

Remark 1: There might be an f (surely less than $n/3$) for which an uptime target for exclusion p is feasible. Nevertheless, the adversary seems to always be able to gain an advantage.

Remark 2: These attacks are not feasible in a synchronous network because the adversary cannot delay the message delivery more than Δ.

5.2.5 Storage requirements for validators

There are many components that contribute to the storage requirements of stakeholders. A short list to be analyzed is the following:

1. All validators store the beacon chain, which contains the block headers of all blocks of all shards. Unless each block has constant size, the bandwidth requirements would be proportional to the validators in the system, which is not scalable. However, even when the block size is constant, storing the block header and the block itself are asymptotically the same [AKW19].

2. In the current implementation, the coins are sharded. This means that a stakeholder most probably will be a validator in multiple shards, even if he is not a major stakeholder. Consequently, each stakeholder must maintain multiple blockchains (shards) and participate in multiple consensus processes, across all shards on which they are assigned.

5.2.6 Shard overload by transactions – Liveness violation

As the state is not partitioned into the shards by design but by the clients themselves, the balanced operation of shards must be examined. We show that the balanced operation of shards with respect to transactions is not guaranteed neither when we assume malicious behavior nor when all clients are rational.

Rational clients. To determine whether this will be the strategy rational clients will follow, we must determine what is their objective, meaning how clients maximize their utility. It is assumed that the low fee and low latency is what determines the clients' utility. According to this, the client is expected to select the shard with the least workload. But the following two problems are identified in this reasoning:

- How does the client know the workload of each shard? Does the client ask the full nodes of each shard for this information? The validators may lie since it is to their benefit to get as many transactions as possible.
- The client that opens an account is expected to do some transactions in the future. If we assume the cross-shard transactions cost double the fee of an intra-shard transaction, the client will choose the shard in which most of its future payees have accounts. By this reasoning, one shard will be constantly overloaded because the benefits of being in that shard will overweight the benefits of changing shards (up to a point). We expect the beacon shard to have almost double the traffic of the other shards. For a more concrete analysis, an evaluation of how important is latency vs fees should be made.

We conclude that rational clients will not maintain transaction-balanced shards.

5.3 Planned Updates

The modulo operation described above will be replaced by a pseudorandom process seeded with a distributed randomly generated number. Specifically, the committee members will run a Distributed Randomness Generation beacon (DRG) which will be secure, assuming $\frac{2}{3}$ of the committee members are honest.[12]

5.4 Proposed Changes

5.4.1 Defense against attack 5.2.1

The planned update 5.3 protects against the attacks described in Section 5.2.1.

5.4.2 Securing honest-majority in each shard (defense against attack 5.2.2)

Reducing the adversarial model assumption to at most $t < 25\%$ of the stake would result in a secure system similar to prior work [KKJG+18, KKJG+16]. Since the DRG will be used to assign the new validators to shards uniformly at random, all shards will have at most $f < 33\%$ with high probability due to Chernoff–Hoeffding bounds: the initial assignment follows the hypergeometric distribution as a fixed number of validators is distributed amongst the shards without replacement. When initially assigning n/k validators to a shard from a pool of n of which $t \cdot n$ are corrupt, the probability that the shard receives at least $f \cdot n/k$ corrupt validators is bounded by:

[12]`https://docs.google.com/document/d/1LGjFXFF3l6OUB-4ryoiiGt2gmrwF6-NIzV5ZOMmMxMA/edit`

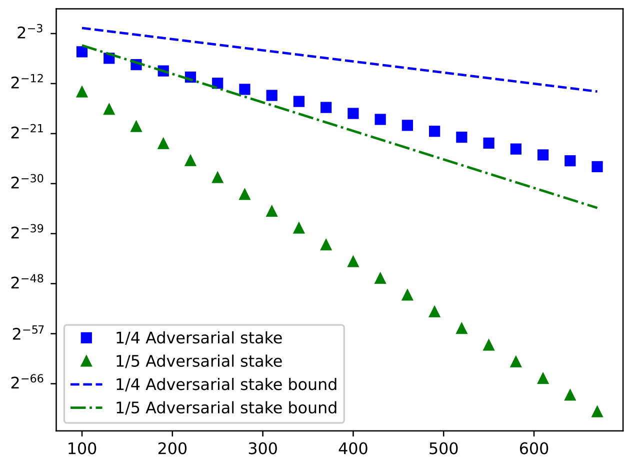

Figure 1: Probability of honest hyper-majority failure with regard to validators per shard and adversarial power. Lines represent Hoeffding bounds, and symbols represent direct calculations.

$$e^{-2((f-t)^2 n/k}$$

That is, the probability that an honest hyper-majority fails to form is negligible in n/k (equiv. n) as long as $f > t$ i.e. we restrict the adversary to control strictly less than $1/3$ of the state. However, it is also useful to perform direct calculation since bounds might not be tight and also because we are interested in concrete probabilities to inform the choice of system parameters.

The calculations presented in Figure 1 show that concrete security is better than what the bounds indicate. The calculations also demonstrate the impact of changing the adversarial stake parameter t from $1/4$ to $1/5$ as the $(f-t)^2$ has considerable impact in the exponent.

Using the current parameter of 250 validators for a single shard as an example, Hoeffding bounds give a probability of failure of 3.1% and 0.0137% for adversarial stakes of $\frac{1}{4}$ and $\frac{1}{5}$ respectively. Direct calculation on the other hand gives a probability of 0.0254% and 2.001e−9 respectively.

At the same time, when assuming $1/4$ adversarial stake, the probability of honest hyper-majority failure might remain significant without also increasing the number of validators. Further investigation is required to fully model this vector, its potential impact and to also consider appropriate mitigation (e.g via slashing).

5.4.3 Maintaining honest and balanced shards in reconfiguration (defense against attack 5.2.3)

The goal is to maintain balanced shards, i.e., shards of approximately the same size, and guarantee honest super-majority in each shard. According to [AS09] these properties are maintained as long as cuckoo's rule is applied polynomially many times. We propose the following for the reconfiguration of shards:

- At each epoch transition, all honest stakeholders that remain active bid on the same slots as in the previous epoch. This is to their benefit because they prefer to remain in the same shard to avoid the overhead of bootstrapping in a new shard.

- The slots that are assigned to the same stakeholder as in the previous epoch are initially assigned to the same shard. The slots that belong to new stakeholders are assigned uniformly at random into one of the shards. Then, the bounded cuckoo's rule is applied to move a constant number of validators to new shards for each new validator that joins.

- We assume the adversary intentionally targets one shard, meaning she will bid on new slots only when they are not assigned to the targeted shard. From the security analysis of [AS09] we know the adversary cannot succeed for a polynomial number of tries. We propose a total reboot once in a while T, e.g., approximately every year, to safely maintain this assumption. In particular, we propose the validators are reassigned uniformly at random into the shards every T blocks. Before the shards reassignment takes place, the validators of the previous epoch reliably broadcast the account balances (current state of their chain) to all stakeholders to ensure data availability.

5.4.4 Defense against uptime attack 5.2.4

Use uptime only for rewards and remove uptime exclusion.

5.4.5 Reducing storage requirement

1. The beacon chain currently stores all block headers with the signatures to determine the uptime. We propose the following: each shard gathers this information in the last block of each epoch and only this is included in the beacon chain – so the calculations are done on each shard and this may relieve the beacon chain. In this way the information on the beacon chain would be linear to the number of validators for each epoch (so amortized it is constant for a polynomial size of epoch with respect to n), while in the current implementation this information is published on the beacon chain for each block.

2. This is an inherent problem in a PoS that shards the coins and highlights a security-storage trade-off. Rational users are expected to try to win

slots that are assigned in the same shard such that they only store and validate transactions in one chain. Eventually this is possible – after some time resharding will allow them to move their slots in one shard. But at the same time this implies that the adversary can target one shard, and therefore possibly violate the security assumptions. As a result, the resharding process must not allow one stakeholder with high concentration of stake to have validators in one shard, but on the other hand it would be beneficial to the protocol to have small stakeholders having validators only in one shard. This is a fine line and demands further analysis to determine if this is possible. We expect this value to depend on how often we reboot the whole system by randomly assigning all validators to new shards (Section 5.4.3).

5.4.6 Transaction balanced shards

We propose using a fee mechanism to incentivize clients to open accounts and transact in different shards, effectively avoiding high traffic in the beacon shard (or any other shard). The core idea is to employ a fee mechanism similar to EIP1559 of Ethereum. For further details and analysis of the fee mechanism, we refer the reader to Section 6.

5.5 Summary of fixes & proposals

- We assume the adversary holds at most 25% of the total stake.
- All active stakeholders are assigned to the same slot as previously. New stakeholders are assigned according to the bounded cuckoo's rule, using the DRG for randomness.
- Once every T blocks, the assignment into shards is done uniformly at random, not following the aforementioned strategy. The exact T should be calculated such that the security-storage trade-off is optimized (Section 5.4.5).
- At the end of each epoch, each shard creates a crosslink where the uptime of each validator in the shard is calculated. The crosslink along with at least 2/3 of the shard's validators' signatures are sent and committed in the beacon chain. This information is used to determine rewards only, but no validator exclusion.
- Data availability/integrity: If we assume only a constant number of validators are new, there is no data availability problem. If we assume more new validators, we need to reliably broadcast the current account balances across all shards to ensure data availability. This step is definitely necessary every T blocks where the shards are reconfigured completely.
- Transaction partition: Incentivize with transaction fees – see next Section for details. In a nutshell, employ the EIP1559 Ethereum fee mechanism (or a variation).

6 Incentives and Rewards

6.1 Current Implementation

At its current state [Lana, Harb], each confirmed block yields a reward of (exactly) 7 ONE[13] for the validators that participated in its production. In particular, the 7 ONE are allocated to each validator whose key signed the block, i.e., who is included in the "commit" bitmap, proportionally to the voting power of the key(s) that signed. Following the initial allocation to validators, each validator further distributes the reward to their delegators, proportionally to each delegator's (delegated) stake, minus a commission fee.

For every slot, the leader broadcasts a bitmap with the signatures of the validators that signed the block. Following, the system awards these validators with a (proportional) reward.

Effectively, all the transaction fees are burned and 7 new ONE are issued per block.

6.2 Current Hazards and Attacks

We assume that all parties are rational and that their utility is equal to their stake, but coalitions of only up to 1/4th of the total stake are allowed.

6.2.1 Incentive incompatible consensus reward allocation

The leader of every slot is responsible (and trusted) for aggregating the received signatures in a bitmap. However, a byzantine leader may manipulate the bitmap to their advantage, while keeping the total amount of included signatures above the threshold of $\frac{2n}{3}$ (n being the number of the shard's validators), as shown in the following example.

Consider an adversary A who controls 33 out of 100 nodes in a shard (without loss of generality, assume the shard consists of 100 validators in total). On expectation, A will be elected as leader for 33 of the 100 block-rounds. Every time A is elected as the leader of a slot, A includes the signatures of the 33 validators it controls, along with 34 signatures of honest validators. Therefore, for the slots where A is the leader, A's reward is $\frac{33}{67} \cdot 7 = 0.49 \cdot 7$ ONE, while the (aggregate) honest parties' reward is $0.51 \cdot 7$ ONE. When an honest party is elected as the slot leader, A follows the protocol and produces the relative signatures; following, the honest leader, who follows the protocol faithfully, includes all available signatures. Therefore, for the 67 honestly-controlled slots, all 100 signatures are included and the expected reward for each such block is $0.33 \cdot 7$ ONE for A and $0.67 \cdot 7$ ONE for the honest parties. To summarize, the total expected reward of the adversary in 100 block-rounds is $(0.49 \cdot 33 + 0.33 \cdot 67) \cdot 7 = 38.28 \cdot 7$ ONE, while the expected (aggregate) reward for the honest parties is $(0.51 \cdot 33 + 0.67 \cdot 67) \cdot 7 = 61.72 \cdot 7$ ONE; thus, A gets more rewards than

[13]`https://github.com/harmony-one/harmony/blob/v4.3.1/staking/reward/values.go#L32`

its fair share (which is 33) to the financial disadvantage of the honest parties (who should have gotten 67). As a result, the stakeholders are not properly incentivized to participate honestly and follow the protocol's specification.

6.2.2 Collusion towards centralization

Since an adversarial leader can exclude signatures from the published bitmap (as shown above), it is possible that rational validators may collude to forcibly exclude other validators. To demonstrate this hazard, consider the following example.

Assume 4 validators, A, B, C, D, each controlling 25% of the epoch's slots. If all validators are honest then, for each block produced in the epoch, each validator receives (on expectation) $0.25 \cdot 7 = 1.75$ ONE [Harb]. However, consider the following scenario. A signals to all parties that it will exclude D's signatures from the bitmap of all blocks that it produces. If the other parties ignore this signal and act honestly, then the rewards of parties A, B, C will be slightly increased compared to the all-honest execution, whereas D's rewards will be slightly reduced. However, if B also follows the same strategy as A, two consequences emerge: i) the rewards of A, B, C further increase, while D's decrease; ii) D's uptime becomes $\frac{1}{2}$, since its signatures are part of blocks produced by only C and D. As a result, due to the $\frac{2}{3}$ the $\frac{2}{3}$ activity threshold, D is excluded from the next epoch. At that point, A, B, C control 33.33% of the epoch's slots each. They can then switch back to the honest protocol and receive $0.33 \cdot 7 = 2.31$ ONE per produced block each, i.e., more than their (original) fair share of 1.75.

We note that this scenario *does not* require a central, actively-controlled coalition between A, B, C, but rather relies on each party's rationality in maximizing their rewards. Also, this can be done on-the-fly, when the leadership schedule is published. For example, the first 3 distinct leaders (e.g., A, B, C) signal to the rest to exclude the fourth. We also note that it is easy to extend this example to multiple leaders with different stake percentages, such that eventually only 3 leaders remain.

6.2.3 Transaction fees

The current macroeconomic policy (i.e., a fixed amount of tokens per block) does not reflect the size or the complexity of published transactions. Therefore, there currently exists no incentive for block creators to validate and include transactions in a block. In particular, it is in the block creators' interest to create empty blocks, receiving the same amount of rewards but also reducing the cost of parsing transactions. Similarly, there is no a direct incentive for users to pay fees for each transaction, as there does not exist a market for space in a block (as is the case with classic cryptocurrencies, like Bitcoin).

Furthermore, without transaction fees the transaction partition into shards is not incentivized financially. In particular, if a market exists for space in a shard, when a shard becomes saturated users' fees (in this shard) would increase, thus users would be incentivized to transact on other shards. As a result, the market mechanism would help balance the load across multiple shards. Without transaction fees, balancing is incentivized only due to high latency, which may be a result of overloaded shards; however, latency is typically a low priority for regular users, compared to fees, so it is not sufficient to incentivize efficient balancing across shards.

6.3 Proposed Changes

The current macroeconomic design of Harmony is "Issuance plus transaction fees set to 441M ONE per year." [Whi] In line with the identified attacks of Section 6.2, this policy has to be revisited. In short, our proposed new policy is: "Issuance plus transaction fees set to *at most* 441M ONE per year."

6.3.1 Fixed Rewards (Issuance)

The principal idea, when trying to mitigate the attacks of Sections 6.2.1 and 6.2.2, is to change the block's rewards, such that the leader receives more rewards proportionally to the number of signatures included in the block's bitmap [KK19].

Specifically, we consider a function $f_s(\cdot)$, which behaves as follows. The output of f_s is a floating point number, which defines the amount of new coins that are rewarded to the leader and the committee members that participate in a block's creation. The input of f_s is an integer x, which expresses the amount of stake of a block's signers (as included in its bitmap) and is equal to the sum of signatures, each weighted by the proportion of stake it corresponds to. Let X be the aggregate stake of all committee members; it now holds that $f_s(X) = I$, where I is a global parameter.

Naturally, to incentivize the leader to include more signatures in the bitmap, the pie should increase as more parties join the sharing. Specifically, the pie should increase as more *stake* backs the block via signatures. The rate of increase of the pie, i.e., the total rewards shared between the leader and committee members, should take into account that the leader may be willing to incur a small short-term loss, aiming for long-term profits (e.g., via the collusion attack of Section 6.2.2); in other words, the leader may act *non-myopically* to some extent. The higher the rate of reward increase, when including all available signatures, the more such behavior is disincentivized. Therefore, we propose that f_s is a *superlinear* function. Of the (many) available superlinear functions, we propose using the square function x^2, which is elegant and intuitive, such that:

$$f_s(x) = \frac{I}{X^2} \cdot x^2 \tag{1}$$

The rewards for all nodes should also be balanced, so the new coins should be allocated to the leader and the validators that sign the block proportionately to their stake. Otherwise, e.g., if the leader receives disproportionately more rewards than the committee members, parties may be incentivized to view-change the leader and become leader themselves. In effect, the issuance rewards are akin to a pie, which is shared by all parties that sign the block, proportionally to their stake. Therefore, if a party P with $\alpha \cdot \hat{X}$ units of stake signs a block, where $\hat{X}$ is the aggregate amount of stake of all parties that sign the block, P's reward from issuance (i.e., the newly-issued coins) is $\alpha \cdot f_s(\hat{X})$.

6.3.2 Transaction Fees

In every cryptocurrency system, validators supply space (in each block) and computational effort (in validating transactions) and users buy said resources by paying fees. Therefore, as described in Section 6.2.3, the transaction fee mechanism should reflect this market dynamic. In effect, if a leader can include a transaction in a block but chooses not to, they should receive fewer rewards.

We will be to focus on transaction gas size. Also, in line with our macroeconomic policy goal, the leader's and committee members' rewards that stem from transaction fees should be upper-bounded, by some limit $\mathcal{T}$. Therefore, when choosing transactions to include in a block, the leader takes into account two limits: i) the block size limit L; ii) the cap on transaction fee rewards $\mathcal{T}$.

Before we proceed with our proposal, let us consider a first elementary option: the transaction fees awarded to the leader and to the committee members are computed as the sum of fees of all transactions included in a block, upper-capped by $\mathcal{T}$, with the excess fees (beyond $\mathcal{T}$) getting burnt. This policy has the benefit of being intuitive, as it follows the paradigm of Bitcoin: i) users compete for space on each block; ii) each transaction defines a (user-chosen) amount of fees and validators order transactions based on the "fee per byte"; iii) in times of congestion, the fees rise as users try to incentivize the validators to prioritize their transaction. iv) miners are incentivized to publish transactions if possible, albeit considering the two limits $L, \mathcal{T}$. However, this policy also has the shortcoming that, when the network is congested, the fees might rise significantly and almost make the system unusable, as has been observed in Bitcoin [Red21] and Ethereum [Got21]. Combined with the upper-limit $\mathcal{T}$, in our setting, this policy would possibly result in situations where a block is mostly empty, except only for a few high-fee transactions which reach the limit $\mathcal{T}$. Naturally, this is undesirable, since it would hurt the efficiency of the system, in terms of both throughput and latency.

To incentivize validators to completely fill the blocks, if possible, we consider a function $f_t(\cdot, \cdot)$. The output of f_t is an integer, which expresses the amount of transaction fees (of the transactions published in a block) which are awarded to the leader and the committee members that sign the block. The input of f_t is two integers: i) the amount of fees defined in all of the block's transactions; ii) the aggregate size of included transactions.

When more transactions are available than can fit in a single block, the leader has to make a choice of leaving some out. To obtain the optimal choice, in terms of capacity, the leader should solve a Knapsack problem, a well-known NP-complete problem. Nonetheless, obtaining a solution which approximates the optimal one, even if not being equally efficient, is good enough for real-world systems. Therefore, the leader should not be expected to necessarily obtain the optimal solution, but rather an approximation is almost good enough. As a result, the rewards that stem from transaction fees should increase significantly as the first few transactions fill the block, but then gradually tend to an upper limit as the block's capacity is reached. The natural candidate functions for such mechanism are *logarithmic* functions. In our case, following our above choice for issuance, we propose the function $\log(x)$.

We now put the two ideas together. The fees that are awarded to the leader and committee members that sign the block are:

$$f_t(F, x) = \min(\mathcal{T}, F \cdot \frac{\log x}{\log L}) \tag{2}$$

where: i) $\mathcal{T}$ is the (global parameter) reward limit for transaction fees; ii) F is the sum of fees defined by all of the block's transactions; iii) L is the block's maximum size;[14] iv) x is the aggregate size of the block's transactions.

This mechanism offers the following interesting properties:

- If the rewards are equal to the defined fees (i.e., no transaction fees are burnt), then the block is (necessarily) completely full.
- As long as a block is not full and the existing aggregate tx fees are not high enough, the leader is incentivized to include transactions in the block; equivalently, users are encouraged to create non-zero fee transactions, to compensate the leader's effort.[15]
- The fees that a rational user will pay are upper-bounded; even if the block is empty except a very small transaction, this transaction's fees should be at most $\mathcal{T} \cdot \log L$ (as any excess fees would be burnt).
- When the network is congested, transaction fees are expected to be bounded by $\frac{\mathcal{T}}{L}$ per fee unit, because: i) users will increase the fees so that leaders prioritize their transactions; ii) defining more than $\frac{\mathcal{T}}{L}$ per fee unit offers no extra incentive to the validators, since the blocks are full (so the maximum reward threshold $\mathcal{T}$ is reached) and the excess fees are burnt. During those periods, users may face longer latency in transaction finality, but also all users are "equal", as richer users are no longer able to push their transactions to the front of the priority queue by paying more fees.[16]

Finally, as with issuance rewards, the reward allocation mechanism should avoid favoring the leader disproportionately, to avoid incentivizing view-changes

[14] There are various metrics that could be used to evaluate block and transaction size, e.g., length (in bytes) or computation requirements (in gas).

[15] If the block is not full and the aggregate tx fees have not reached the threshold, including zero-fee transactions would increase x and, thus, the block's rewards. Nonetheless, presumably this increase is marginal and is not sufficient to cover the leader's validation operation cost.

[16] If validators assign a large utility in producing small blocks, they might be incentivized to include a small transaction with excess fees (enough to reach the limit even after a large part of them is burnt) over producing a nearly-full block. We assume that validators value maintaining the system's liveness more than slightly reducing their operational costs though (i.e., by validating fewer transactions).

by the committee members. Therefore, again the rewards that stem from transaction fees should be allocated to the leader and each member that signs the block linearly to each validator's stake.

6.4 Summary of Fixes & Proposals

We propose a change in the macroeconomic policy of Harmony. The new policy should be "Issuance plus transaction fees set to *at most* 441M ONE per year.".

Each block defines a level of rewards R, which is at most 7 ONE. These rewards are split in two parts:

1. issuance R_I, with an upper limit I;
2. transaction fees R_T, with upper limit $\mathcal{T}$.

Therefore, it holds that $R = R_I + R_T \leq I + \mathcal{T} = 7$ ONE.

The issuance rewards are computed as $R_I = f_s(x) = \frac{I}{X^2} \cdot x^2$, where X is the aggregate amount of stake of all committee members (plus the leader) and x is the amount of stake of the leader and the committee members that sign the block (according to the block's bitmap).

The transaction fee rewards are computed as $R_T = f_t(x) = \min(\mathcal{T}, F \cdot \frac{\log x}{\log L})$, where F is the aggregate fees defined by all transactions in the block, L is the block's size limit, and x is the aggregate size of the block's transactions.

The block's rewards R are allocated to the leader and the committee members that sign the block proportionally to each party's stake percentage across all committee members. Therefore, if a party P controls $\alpha \cdot X$ units of stake and signs a block that offers rewards R, P's reward is $\alpha \cdot R$.

Macroeconomic parameters and expected validators' rewards. There are two principal attributes of the reward mechanism that should be taken into account when choosing the parameters (e.g., $f_s, f_t, I, \mathcal{T}$ etc). First, the mechanism should not generate more rewards than needed, as this could lead (under certain market conditions) to excess supply and, consequently, a drop in the token's price. In the above mechanism this is guaranteed by the cap of 7 ONE per block. Second, the expected rewards per time period (e.g., per week) should be predictable and stable, which would help validators estimate the income stream for a foreseeable future. Ideally, the block rewards should be as close to 7 ONE as possible, while offering proper incentives, as discussed above. However, this is not always possible; for example, if say $I = 2$, if no trasactions exist (e.g., due to inactivity), the validators will receive at most 2 ONE, which is a significant reduction from the maximum target. Therefore, the choices of the reward mechanism's parameters should be informed by historical data from Harmony's network and the expected behavior of its users. In addition, a mechanism for updating the macroeconomic principles of the system should be put in place (e.g., as in [Zin21]), to enable policy changes when the environment and the expectations change.

6.4.1 Selecting T & I

The choice of I and $\mathcal{T}$ presents a tradeoff between enhancing safety or liveness respectively. Larger I increases the incentives for committee members to sign a block and for the leader to include such signature. Larger $\mathcal{T}$ increases the freedom of users to increase fees and compete for space in a block, as well as possibly increases the number of transactions that a leader is incentivized to publish in a block if the network is not congested (which would result in reduced latency in transaction publishing). Therefore, a future design could allow some flexibility in $I, \mathcal{T}$, e.g., increasing $\mathcal{T}$ in periods of congestion. Nonetheless, I and $\mathcal{T}$ should always be non-zero, to provide some incentive against all of the hazards of Section 6.2.

In order to better inform the selection of $I, \mathcal{T}$ we have conducted a simulation, using historical data from Harmony's ledger. Specifically, we estimated the expected rewards, under the proposed scheme and for different options for $I, \mathcal{T}$, with respect to the blocks of shard 0 during epoch 882. The result is showcased in Figure 2, while the average rewards per block are as follows:

- $(I, \mathcal{T}) = (2, 5) : 2.872638949492717$
- $(I, \mathcal{T}) = (3, 4) : 3.8265000261150077$
- $(I, \mathcal{T}) = (4, 3) : 4.769173076531937$
- $(I, \mathcal{T}) = (5, 2) : 5.678249152025853$
- $(I, \mathcal{T}) = (6, 1) : 6.526517850369548$

The historical data show that committee members typically sign all blocks when elected, with only a few exceptions. Additionally, blocks are typically not full (only 2,081 of 32,768 blocks were at least 80% full) and transaction fees are typically not high (only 1,262 blocks offered full transaction rewards when $T = 5$). Under these circumstances, if the priority is on avoiding spam transactions (i.e., increasing $\mathcal{T}$), then the blocks rewards and, consequently, the validator's revenue is expected to drop by even 50%. Instead, if the priority is on retaining the validators' revenue and further incentivizing them to participate in block signing, then the block rewards will not face a significant reduction (as showcased in the case $(I, \mathcal{T}) = (6, 1)$ above). Finally, a dynamic adjustment of $I, \mathcal{T}$ could be employed in the future; for instance, $\mathcal{T}$ could be initally kept low, while transactions are few, and increase after a series of epochs when blocks are almost full.

7 Penalties and Slashing

Slashing not applied. As of release 4.3.1, the below slashing logic is implemented[17] but is not active, as it is commented out[18]. Therefore, there currently

[17] `https://github.com/harmony-one/harmony/blob/v4.3.1/staking/slash/double-sign.go`

[18] `https://github.com/harmony-one/harmony/blob/v4.3.1/consensus/leader.go#L268`

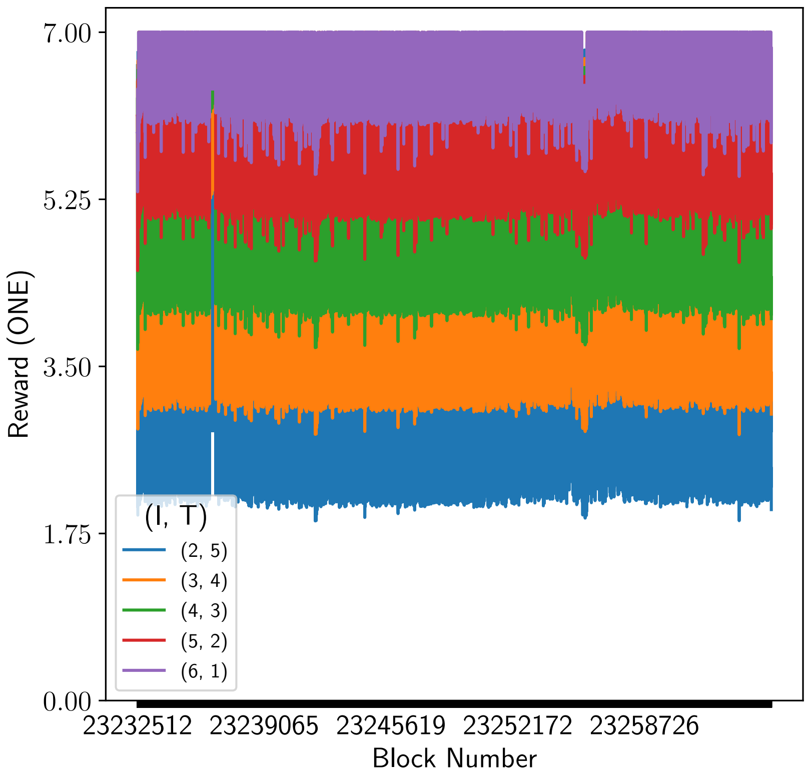

Figure 2: The rewards of each block in shard 0 of Harmony's ledger during epoch 882 (block numbers 23,232,512 – 23,265,279), under our proposed reward mechanism and for different values of $I, \mathcal{T}$.

exist no penalties for misbehavior, although, in our understanding, the below described mechanism is the *intended* one to be enforced at some point in the future.

7.1 Current Implementation

Slashing occurs when a party double-signs [Harb], i.e., signs two different blocks with the same height and view ID. When a validator V double-signs, a percentage of the stake that V manages (i.e., their own and the stake delegated to them) is slashed. The slashing occurs after a party issues an "accusing" transaction, which contains a proof of the double-sign.

The slashing percentage is computed as the sum of the percentages of voting power of all keys that reportedly double-signed (during the same accusation), at a minimum value of 2%. Slashing is applied proportionately to every party that had delegated to V, [19] during the epoch when the double-signing occurred. If a party's stake is not sufficient to cover for the slashing, e.g., if it undelegated in the meantime, the party is charged as much as possible.[20] Finally, the offending party is blacklisted[21], after which point it cannot become a validator again, neither can it be slashed for the same or other offenses.

Let us provide intuition via an example. Consider a validator V who, on epoch e, manages 100 coins, which is equal to 3% of the total participating stake. Of these coins, 10 are owned by V, 50 are owned by party A and 40 by party B, who both delegate their staking rights to V. During epoch e, V double-signs, as do 2 other validators, V_1, V_2, each with 3.5% of the total stake respectively. A reporting party collects proofs of double-signing for all the parties V, V_1, V_2 and issues an accusation. Thus, the percentage of slashed stake *for each of the validators* V, V_1, V_2 is 10%. Specifically for V, 10 of V's managed coins are slashed, of which 1 is owned by V, 5 are owned by A, and 4 are owned by B.

Slashing takes effect even if some stakeholders redelegate their stake in the meantime. In the previous example, assume that the accusation is made on epoch $e + 2$, whereas B changes its delegation preference on epoch $e + 1$, redelegating from V to another validator V'. B will still incur a slashing penalty of 4 coins, for the past offending actions of its then-delegate (V).

Finally, the reporting party is rewarded for making a valid accusation. Of the slashed stake, half is burnt, while the other half is credited as reward to the reporting party [Harb]. Additionally, there is no upper time limit within which a double-sign instance can be reported, thus a party can make an accusation for a double-sign instance that relates to any past epoch.

Note that any slashing mechanism must take into account that a validator can self-delegate instead of directly staking her coins. This is a kind of sybil attack and could be of benefit to the validator if e.g. the slashing mechanism is more lenient against delegated coins than against directly staked ones.

7.2 Current Hazards and Attacks

7.2.1 Colluding validators

Consider a validator V who manages s units of stake. s is split in two parts, $s = s_V + s_d$, where: i) s_V is owned by V; ii) s_d is the stake delegated to V by other parties.

At some point, V performs a double-signing, which incurs slashing percentage x. Therefore, $x \cdot s$ of V's managed stake is slashed, of which $x \cdot s_V$ is owned by V and $x \cdot s_d$ is owned by V's delegators.

Let V' be the reporting party that publishes the proof of V's double-signing. V' receives, as reporting reward, $R = \frac{x \cdot s}{2}$ of (the slashed) stake.

[19]`https://github.com/harmony-one/harmony/blob/v4.3.1/staking/slash/double-sign.go#L503`

[20]`https://github.com/harmony-one/harmony/blob/v4.3.1/staking/slash/double-sign.go#L337`

[21]`https://github.com/harmony-one/harmony/blob/v4.3.1/staking/slash/double-sign.go#L512`

Now, consider the case where V, V' are colluding[22]. Specifically, V, V' share the reporting reward R. If R is higher than V's penalty, then V is incentivized to double-sign (as long as it is V' who reports it and gets the reward). Specifically, V is incentivized to perform the attack if the following holds:

$$\begin{aligned} R &> x \cdot s_V \Rightarrow \\ \frac{x \cdot s}{2} &> x \cdot s_V \Rightarrow \\ \frac{x \cdot (s_V + s_d)}{2} &> x \cdot s_V \Rightarrow \\ s_d &> s_V \end{aligned} \tag{3}$$

Therefore, the attack is viable if the validator's self-owned stake is less than the stake delegated to them. As a result, via this attack V incurs a loss of $x \cdot s_d$ stake on its delegators and obtain a profit $x \cdot \frac{s_d - s_V}{2}$.

Current State: As of October 2021, all of Harmony's active validators[23] satisfy the above inequality, i.e., no validator owns more stake than their aggregate delegated stake. The highest percentage of self-stake (over all stake managed by the validator) is by addresses *one1y568...sgrt* and *one1xrlz...esgc*, who own 44.69% and 43.32% of their total stake respectively; the lowest percentage is observed by addresses *one14q6l...t7qj, one14l4l...mgdc, one1upv5...rcl5, one1aha9...959z* who own less than 0.01% of their total controlled stake.

7.2.2 Posterior double-signing

Assume a validator V, who is elected in the committee of epoch e. During e, V is a leader for some slot s and creates (and signs) a block B.

At a later epoch $e' > e$, V creates a block $B' \neq B$ for the same slot s of epoch e, i.e., which is conflicting with B. This is feasible if long-range attacks are not mitigated, as V could gain a majority of the previous keys (buy them or later in time activate previously inactive keys). As a result, the blocks B, B' and the corresponding signatures are proof of double-sign, for which V should be slashed. The slashing penalty is computed over V's managed stake on the epoch which corresponds to the double-signed block, in this case, e.

However, it is possible that, on epoch e', a stake shift has occurred. Therefore, V's delegators on epoch e may no longer be in the system, e.g., may have sold their stake. Similarly, on epoch e', V may not control the same (or even any) amount of stake that they had controlled on e. Consequently, V can double-sign a past block and not get penalized for it.

Note: Long-range attacks induce even harsher security vulnerabilities. This section describes their effect on slashing.

[22]In this case, collusion does not necessarily imply that V, V' are distinct users. Instead, V, V' could be two pseudonymous identities controlled by the same malicious party.

[23]Data obtained from: `https://staking.harmony.one/validators/mainnet/one1p2e0a0806jv8tqr37k7c8k67zgfjwtzpf9apv2` [12 October 2021]

7.2.3 Redelegation locking period

As described in Section 3.1.3, after a user initiates the undelegation of its coins from a specific delegate, two time limits may apply: i) the coins stay locked for 7 epochs and are then completely undelegated; ii) the coins are redelegated to another delegate without delay, i.e., from the beginning of the next epoch.

Considering the attacks of the Sections 7.2.1 and 7.2.2, a malicious validator V could act as follows:

1. V does not directly lock funds in its key, but rather delegates them from another key that they control.
2. At the end of epoch e, V undelegates all of its self-staked tokens.
3. At the beginning of epoch $e + 7$, the undelegation takes effect, so $s_V = 0$; at this point, V performs posterior double-signing, creates a proof of a slashable offense for epoch e, which V' publishes.

Consequently, V incurs no slashing penalty themselves, since $s_V = 0$. However, V's delegators do incur a slashing penalty, which is awarded to the malicious party as described above.

Observe that V can initiate the undelegation at the end of epoch e. So, even if V's delegators observe this action and try to react by also undelegating, they will do so on epoch $e + 1$. Therefore, on epoch $e + 7$, when the proof of V's double-signing is published, their coins will still be locked (and consequently slashed).

7.2.4 Slashing reward front-running

Consider a party V, who wishes to report a double-signing. V creates an accusation transaction that contains the double-sign proof, i.e., the offending blocks and signatures, and sends it on the network for publication.

The transaction is received by all parties, including the validator V_A, who is the leader of the slot after V makes the accusation. Since the double-sign proof is included in the transaction in plaintext, V_A retrieves it and creates an accusation transaction of their own, which awards the reward to V_A's account. Therefore, V_A effectively front-runs V in obtaining the accusation reward.

Note: This is an instance of a larger family of attacks, directly related to "Miner Extractable Value" [Eth].

7.2.5 Blacklist resource requirements and replay protection

The full nodes are required to maintain a blacklist of offending validators and leaders. This blacklist serves a dual purpose. First, it is intended to prevent future or repeated offenses by the same parties. Second, it acts as a replay protection mechanism, such that a party is not slashed twice for the same offense; in particular, when an accusation is made for a validator, slashing takes effect only if the validator is not already in the blacklist.

Regarding the first purpose, considering the permissionless nature of the system, where parties can join or leave at any time and also create pseudonymous, unlinkable identities, these records have little value. Specifically, a malicious party can simply create multiple identities/keys, perform an offense with one of them and then "burn" it, i.e., never reuse it. More importantly, the full nodes of the system are required to consume storage and computational resources in maintaining these records. Therefore, considering the ability of parties to use "one-off" identities, these records are expected to be ever-increasing.

Regarding the second purpose, the current blacklist implementation is rather rudimentary, storing only the identity (i.e., key) of a slashed validator.[24] Therefore, if the validator performs multiple different offenses, they will be slashed only for one of them (the first that is reported).

7.3 Proposed Changes

7.3.1 Colluding validators

The following are the primary reasons why the attack of Section 7.2.1 is possible:

1. there exists no identity check for validators and double-sign reporters;

2. a validator's self-owned stake may be less than the stake delegated to them;

3. slashing is applied both on the offending validator's self-owned stake and the stake delegated to them, at exactly equal proportions;

4. the reporter's reward is exactly equal to $\frac{1}{2}$ of the total slashed stake.

The above allow an attacker to control both a malicious validator and a reporter and, by double-signing and reporting the offense, to claim more reporting rewards (from its delegators) than its slashed self-owned stake.

Of the above points, the first is unavoidable in decentralized, permissionless systems. Therefore, it is impossible to prevent the collusion of validators and reporters, neither to identify a single adversary controlling multiple identities that act on either role.

If the number of validators is significantly smaller than regular users and the wealth distribution across all users is wide, point 2 also seems unavoidable. A major advantage of delegation is allowing the majority of users to be offline, while a small, dedicated group acts on their behalf. In a system where wealth is not concentrated around a small group, which would also act as validators, it is expected that the majority of stake will be delegated.

Therefore, any actions to mitigate the attack of Section 7.2.1 can focus on points 3 and 4. The end goal should be that inequality 3 does not hold, i.e., the reporting reward should be less than the offending validator's slashed self-owned stake. The following options exist:

[24]`https://github.com/harmony-one/harmony/blob/v4.3.1/staking/slash/double-sign.go#L163`

1. increase the amount of self-owned stake that is slashed;
2. reduce the reporting reward, which can be achieved in two ways:
   (a) reduce the amount of delegated stake that is slashed;
   (b) increase the amount of slashed stake that is burnt.

Options 1 and 2.a imply a higher responsibility of the validator, who acts in the context of the consensus protocol, compared to its delegators, who simply choose a validator. Option 2.b implies a same level of responsibility of the validators and their delegators.

Before making a proposal, let us now reconsider the computation of the slashing percentage. Currently, it depends on the concurrent double-signing instances, which are reported at the same time. Consider a double-signing instance, for a block that pertains to a (time) slot s and which is reported at time $t > s$. Typically, t is either very close to s or very far from it; in the former case, the validator double-signs at the moment when it should act honestly and the proof is made available immediately (so the reporting is made a few slots later); in the latter case, the validator performs posterior double-signing, so the proof is published at a (relatively) much later point in time. In both cases, if multiple parties double-sign, it is uncommon that these proofs will be made available at the same time. Therefore, it is unclear what is the benefit of this mechanism.

To conclude, our proposal is a combination of options 1 and 2.a as the mechanism for slashing and rewarding the offense reporting.

First, the reporting reward remains equal to $\frac{1}{2}$ of the total slashed stake, while the other half is burnt.

The validator's self-owned slashed stake is computed taking into consideration two facts: i) double-signing may be the result of software bugs instead of malicious behavior; ii) posterior double-signing. Therefore, we propose that the validator's *self-owned* stake that is slashed is the minimum of: i) 50% of their stake owned during the epoch which pertains to the double-signing; ii) their entire stake at the epoch when the reporting takes place (e.g., if many epochs have passed and the validator has undelegated a large part of its stake).

The aggregate slashed stake of the delegators (during the epoch which pertains to the double-signing) is at most 80% of the leader's self-owned slashed stake. Specifically, the nominal aggregate penalty for all delegators is 80% of the leader's slashed stake. Each delegator is charged a part of the aggregate penalty which is proportional to their stake (over all delegated stake). If the delegator's stake is not sufficient to cover their penalty (due to posterior double-signing), their entire stake is slashed. Note that in any case, the slashed amount of the leader is greater than the total amount slashed from all its delegates. Therefore, the leader can never gain from launching a colluding attack (i.e., penalizing his own delegators later – see Section 7.2.1).

To make the mechanism clear, let us provide an example. A validator V owns 100 coins. Parties A, B, C delegate $100, 70, 30$ coins respectively to V. We have the following scenarios, for a double-signing made by V regarding a slot in epoch e:

- The proof is published in epoch e; the slashing stake is as follows: V 50 coins, A 20 coins, B 14 coins, C 6 coins. This is because the 80% of 50 coins is 40 coins = 20+14+6 coins.
- The proof is published in epoch $e' \gg e$, during which V controls 30 coins, A controls 100 coins, B, C control 6 coins; the slashing stake is as follows: V 30 coins, A 12 coins, B 6 coins, C 3 coins. In this case, V does not own 50 coins hence the maximum slashing amount is 30 coins. Similarly to the previous case, the delegators are slashed proportionally to an amount that sums to 80% of 30 coins which is 24 coins.

Remarks:

- A validator's self-owned stake reflects their (economic) security guarantees. Therefore, this is a crucial metric that users should take into account when choosing their delegate. If a validator does not present a high amount of self-owned stake (as is the current state of the system, cf. Section 7.2.1), then the economic argument that pertains to slashing is not strong.
- Other parts of the system (e.g., the committee selection mechanism) could also aim at increasing the amount of self stake (e.g., by prioritizing validators with higher amounts of self-owned stake when delegating), in order to increase the economic security guarantees.

7.3.2 Slashing reward front-running

Front-running by the block creators and processors is a fundamental problem in blockchain systems. The particular case of rewards with respect to slashing is also a well-known issue (e.g., in Ethereum[25]). Although this problem is not a direct threat to the system's security or usability, it can be frustrating for users. To minimize the effects of this problem, we highlight three options.

First, the reporting reward may be awarded to the leader who includes a slashing proof in a block. In particular, regardless which user creates the proof and/or submits the transaction that records the accusation, the slashing reward is given to the leader who publishes the transaction in a block they create. Naturally, this choice means that regular users have no incentives to report double-signing instances, apart from altruistic behavior. Nonetheless, leaders, who are a non-negligible set of participants, have such incentive and are expected to do such reporting work. Additionally, this design has the benefit of being easily implementable and of increasing the leaders' potential revenue.

Second, a commit-reveal scheme could be employed. In particular, the user that makes the accusation first commits to the proof and, at a later point,

[25] `https://github.com/ethereum/consensus-specs/issues/1631`

reveals it. Such schemes are a common mitigation against front-running attacks [EMC19]. However, it does increase the complexity and introduce a performance overhead. Specifically, the latency would increase (between the time an offense takes place and when the offender is slashed), while also reporting parties would need to pay more transaction fees for publishing the proof.

Third, no action can be taken, but rather the problem should be communicated to the users in a clear manner. In particular, a user should know (and expect) the possibility that they may not receive the reporting reward, even if they create an accusing transaction correctly.

We propose that the first option is adopted. The second option introduces a level of complexity that we believe is unwanted, at this instance. The third option, depending on the communication strategy, possibly does not resolve the issue, as some users may not become aware of the intricate details of the blockchain protocol. Therefore, the first option offers an adequate balance between usability and security.

7.3.3 Replay protection and the blacklist

Our proposal is to enhance the blacklist with information about each specific slashing offense. Instead of storing only the offender's information, the blacklist should also include the details of the event, i.e., the epoch, slot, and shard during which the double-signing occurred. In addition, if a party makes multiple offenses, e.g., multiple double-signings across different slots, the slashing penalty should be applied for each one.

As elaborated in Section 7.2.5, the blacklist offers some security. It remains an open question what is the optimal way to prevent replay attacks (on double-signing proofs). The blacklist is a stateful scheme, in the sense that it requires storage and computational resources to the always spent by the system's maintainers. An interesting line for future research is possible stateless solutions. Specifically, slashing proofs that can be verified without any historical information, but rather only within the context of a transaction and/or block and/or epoch.

Acknowledgments

This work was funded by Harmony. We would like to thank Pyrros Chaidos, Lefteris Kokoris-Kogias, Dionysis Zindros, Rongjian Lan, Orfeas Stefanos Thyfronitis Litos, Dimitris Lamprinos, and Aleksis Brezas for their valuable discussion and feedback, as well as their impactful contribution to this work.

References

[ACDCKK18] Elli Androulaki, Christian Cachin, Angelo De Caro, and Eleftherios Kokoris-Kogias. Channels: Horizontal scaling and confidentiality on permissioned blockchains. In *European Symposium on Research in Computer Security*, pages 111–131. Springer, 2018.

[AKW19] Georgia Avarikioti, Eleftherios Kokoris-Kogias, and Roger Wattenhofer. Divide and scale: Formalization of distributed ledger sharding protocols. *CoRR*, abs/1910.10434, 2019. URL: `http://arxiv.org/abs/1910.10434`, `arXiv:1910.10434`.

[AS09] Baruch Awerbuch and Christian Scheideler. Towards a scalable and robust dht. *Theory of Computing Systems*, 45(2):234–260, 2009.

[BKKS20] Lars Brünjes, Aggelos Kiayias, Elias Koutsoupias, and Aikaterini-Panagiota Stouka. Reward sharing schemes for stake pools. In *IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020*, pages 256–275. IEEE, 2020. `doi:10.1109/EuroSP48549.2020.00024`.

[Bon05] Dan Boneh. BLS short digital signatures. In Henk C. A. van Tilborg, editor, *Encyclopedia of Cryptography and Security*. Springer, 2005. `doi:10.1007/0-387-23483-7\_35`.

[CL02] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002. `doi:10.1145/571637.571640`.

[DLS88] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, apr 1988. `doi:10.1145/42282.42283`.

[EMC19] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. Sok: Transparent dishonesty: Front-running attacks on blockchain. In Andrea Bracciali, Jeremy Clark, Federico Pintore, Peter B. Rønne, and Massimiliano Sala, editors, *Financial Cryptography and Data Security - FC 2019 International Workshops, VOTING and WTSC, St. Kitts, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*, volume 11599 of *Lecture Notes in Computer Science*, pages 170–189. Springer, 2019. `doi:10.1007/978-3-030-43725-1\_13`.

[Eth] Ethereum. Miner extractable value (mev). URL: `https://ethereum.org/en/developers/docs/mev/`.

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015. `doi:10.1007/978-3-662-46803-6\_10`.

[Got21] Will Gottsegen. Ethereum's fees are too damn high, 2021. URL: https://www.coindesk.com/tech/2021/11/22/ethereums-fees-are-too-damn-high/.

[Hara] Harmony. Consensus. URL: https://docs.harmony.one/home/general/technology/consensus.

[Harb] Harmony. Effective proof-of-stake. URL: https://docs.harmony.one/home/general/technology/effective-proof-of-stake.

[Harc] Harmony. Fbft. URL: https://medium.com/harmony-one/fast-byzantine-fault-tolerance-8bd24eb5e65d.

[Hard] Harmony. Harmony whitepaper. URL: https://harmony.one/pdf/whitepaper.pdf.

[Hare] Harmony. Shard assignment. URL: https://docs.harmony.one/home/network/validators/definitions/validator-keys-and-bids/shard-assignment.

[Harf] Harmony. Validator, bls key, instance. URL: https://docs.harmony.one/home/network/validators/definitions/validator-keys-and-bids.

[KK19] Eleftherios Kokoris-Kogias. Robust and scalable consensus for sharded distributed ledgers. *IACR Cryptol. ePrint Arch.*, 2019:676, 2019.

[KKJG+16] Eleftherios Kokoris Kogias, Philipp Svetolik Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Alexander Ford. Bitcoin meets collective signing. In *37th IEEE Symposium on Security and Privacy*, number POST_TALK, 2016.

[KKJG+18] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 583–598. IEEE, 2018.

[Lana] Rongjian Lan. The definitive guide to harmony open staking. URL: https://medium.com/harmony-one/the-definitive-guide-to-harmony-open-staking-6c78976a7d63.

[Lanb] Rongjian Lan. Introducing harmony's effective proof-of-stake (epos). URL: https://medium.com/harmony-one/introducing-harmonys-effective-proof-of-stake-epos-2d39b4b8d58.

[Lanc] Rongjian Lan. Restore 7 epoch locking period with redelegation. URL: https://medium.com/harmony-one/restore-7-epochs-locking-period-with-redelegation-6b15c47c532c.

[MS83] Roger B. Myerson and Mark A. Satterthwaite. Efficient mechanisms for bilateral trading. *Journal of Economic Theory*, 29(2):265–281, April 1983. URL: `https://ideas.repec.org/a/eee/jetheo/v29y1983i2p265-281.html`.

[Red21] Jamie Redman. Sky high bitcoin and ethereum fees: While prices jump the cost to transfer follows suit, 2021. URL: `https://news.bitcoin.com/sky-high-bitcoin-and-ethereum-fees-while-prices-jump-the-cost-to-transfer-foll`

[Rou16] Tim Roughgarden. *Twenty lectures on algorithmic game theory.* Cambridge University Press, 2016.

[Whi] Nick White. Harmony's new tokenomics. URL: `https://medium.com/harmony-one/harmonys-new-tokenomics-bcdac0db60d7`.

[Zin21] Dionysis Zindros. Soft power: Upgrading chain macroeconomic policy through soft forks. In Matthew Bernhard, Andrea Bracciali, Lewis Gudgeon, Thomas Haines, Ariah Klages-Mundt, Shin'ichiro Matsuo, Daniel Perez, Massimiliano Sala, and Sam Werner, editors, *Financial Cryptography and Data Security. FC 2021 International Workshops - CoDecFin, DeFi, VOTING, and WTSC, Virtual Event, March 5, 2021, Revised Selected Papers*, volume 12676 of *Lecture Notes in Computer Science*, pages 467–481. Springer, 2021. `doi:10.1007/978-3-662-63958-0\_36`.

[ZMR18] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: Scaling blockchain via full sharding. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 931–948. ACM, 2018. `doi:10.1145/3243734.3243853`.