Harmony

Technical Whitepaper

Harmony Team

Version 2.0

1. Introduction

Since the publication of the Bitcoin whitepaper in 2008, the concept of blockchain has spread across the world. While decentralized money and applications are becoming well-publicized ideas, design limitations have challenged the core aspiration of Bitcoin. The original Bitcoin blockchain was designed as a peer-to-peer payment system [13] that allows people to transfer value without intermediaries like banks or payment processors. However, as Bitcoin gained popularity, its performance bottleneck became evident due to its limited throughput of ~7 transactions per second (TPS), and its cost as a payment system became prohibitively expensive.

In 2014, Buterin et. al. [27] proposed a new blockchain infrastructure called Ethereum, which enabled developers to create various kinds of blockchain applications using “smart contracts.” However, Ethereum didn’t solve the scalability problem and, with its ~15 TPS, failed to support high-throughput applications such as gaming or decentralized exchanges.

Given Ethereum and Bitcoin’s performance limitations, many blockchain projects proposed various solutions [3,4,5,6,7,8,9,10,24,25] that attempt to increase transaction throughput. Various blockchains [3,4,5,6,24,25] proposed to replace Proof-of-Work (PoW) consensus with Proof-of-Stake (PoS) consensus. Other blockchains like EOS use Delegated Proof of Stake (DPoS), where block proposers are elected by voting rather than by an on-chain algorithmic process. Projects like IOTA replaced the chain-of-blocks data structure with a DAG (Directed Acyclic Graph) data structure, which breaks the limitation of sequential processing of transactions.

However, these proposed solutions cannot make significant performance gains without sacrificing other critical aspects, such as security and decentralization. The scalability solution that both preserves security and decentralization is sharding, which creates multiple groups (i.e. shards) of validators and lets them process transactions concurrently. As a result, the total transaction throughput increases linearly as the number of shards grows. Zilliqa [12] was the first public blockchain that proposed to address the scalability problem with sharding. However, Zilliqa's sharding approach falls short in two ways. First, it does not divide the storage of blockchain data (state sharding). This prevents machines with limited resources from participating in the network, thus curtailing decentralization. Second, Zilliqa’s sharding process is susceptible to a single-shard takeover attack due to its reliance on PoW as its randomness generation mechanism.

We introduce Harmony, the next generation sharding-based blockchain that is fully scalable, provably secure, and energy efficient. Harmony addresses the problems of existing blockchains by combining the best research results and engineering practice in an optimally tuned system. Specifically, Harmony makes breakthroughs in following aspects:

- **Fully Scalable:** Harmony shards not only the network communication and transaction validation like Zilliqa, but also shards the blockchain state. This makes Harmony a fully scalable blockchain.

- **Secure Sharding:** Harmony's sharding process is provably secure thanks to the distributed randomness generation (DRG) process which is unpredictable, unbiaseable, verifiable and scalable. Harmony also reshards the network in a non-interruptive manner to prevent against slowly adaptive byzantine adversaries.
- **Efficient and Fast Consensus:** Unlike other sharding-based blockchains which require PoW to select validators, Harmony is based on PoS and thus energy efficient. Consensus is reached with a linearly scalable BFT algorithm that's 100 times faster than PBFT.
- **Adaptive-Thresholded PoS:** The threshold of stakes required for a node to join the network is adjusted based on the volume of total staking in a way that malicious stakers cannot concentrate their power in a single shard. Moreover, the threshold is low enough so that small stakers can still participate in the network and earn rewards.
- **Scalable Networking Infrastructure:** With RaptorQ fountain code, Harmony can propagate blocks quickly within shards or across network by using the Adaptive Information Dispersal Algorithm. Harmony also adopts Kademlia routing [37] to achieve cross-shard transactions that scale logarithmically with the number of shards.
- **Consistent Cross-Shard Transactions:** Harmony supports cross-shard transactions with shards directly communicating with each other. An atomic locking mechanism is used to ensure the consistency of cross-shard transactions.

By innovating on both the protocol and network layers, Harmony provides the world with a scalable and secure blockchain system that is able to support the emerging decentralized economy. Harmony will enable applications which were not previously feasible on blockchain, including high-volume decentralized exchanges, interactive fair games, Visa-scale payment systems, and Internet-of-Things transactions. Harmony strives to scale trust for billions of people and create a radically fair economy.

2. Consensus Mechanism

The consensus protocol is a key component of any blockchain. It determines how securely and quickly blockchain validators[1] reach consensus on the next block. The first blockchain consensus protocol which powers Bitcoin is Proof-of-Work (PoW) consensus. PoW is a process whereby miners race to find the solution to a cryptographic puzzle—the winner gets the right to propose the next block and earns some token rewards. PoW's security assumption is that more than 50% of the hashing power is controlled by honest nodes. With such an assumption, the rule for consensus is that the longest chain will be the canonical one, and thus PoW consensus is also called *chain-based consensus*.

Another type of consensus protocol, one which has been researched for more than two decades in academia, is called PBFT (Practical Byzantine Fault Tolerance) [14]. In PBFT, one node is elected as the "leader," while the rest of the nodes are "validators." Each round of PBFT consensus

[1] The machines that support the blockchain network by validating transactions and reaching consensus.

involves two major phases: the prepare phase and the commit phase. In the prepare phase, the leader broadcasts its proposal to all of the validators, who in turn broadcast their votes on the proposal to everyone else. The reason for the rebroadcasting to all validators is that the votes of each validator need to be counted by all other validators. The prepare phase finishes when more than $2f+1$ consistent votes are seen, where f is the number of malicious validators, and the total number of validators plus the leader is $3f+1$. The commit phase involves a similar vote counting process, and consensus is reached when $2f+1$ consistent votes are seen. Due to the rebroadcasting of votes among validators, PBFT has $O(N^2)$ communication complexity, which is not scalable for a blockchain system with hundreds or thousands of nodes.

As an improvement on PBFT [14], Harmony's consensus protocol is linearly scalable in terms of communication complexity, and thus we call it Fast Byzantine Fault Tolerance (FBFT). In FBFT, instead of asking all validators to broadcast their votes, the leader runs a multi-signature signing process to collect the validators' votes in a $O(1)$-sized multi-signature and then broadcast it. So instead of receiving $O(N)$ signatures, each validator receives only one multi-signature, thus reducing the communication complexity from $O(N^2)$ to $O(N)$.

The idea of using $O(1)$-sized multi-signature is inspired by ByzCoin's BFT [15] which uses the Schnorr signature scheme for constant-sized multi-signature aggregation and forms a multicast tree among validators to facilitate the message delivery. However, a Schnorr multi-signature requires a secret commitment round, which leads to a total of two round-trips for a single multi-signature. Harmony improves upon that by using BLS (Boneh–Lynn–Shacham) multi-signature [28], which only requires one round-trip. Therefore, FBFT is at least 50% faster than ByzCoin's BFT. Besides, Harmony adopts RaptorQ fountain code to speed up the block broadcasting process (discussed in §6.2). The fountain code broadcasting technique also avoids a security issue in ByzCoin's original tree-based multicasting design [16,17].

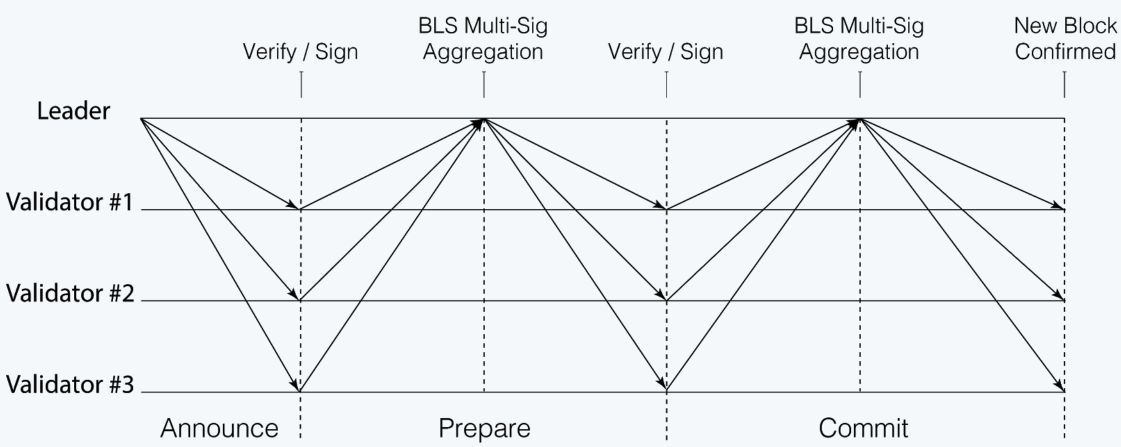

Figure 1. Network communication during a single round of consensus.

Specifically, Harmony's FBFT consensus involves the following steps:

1. The leader constructs the new block and broadcasts the block header to all validators. Meanwhile, the leader broadcasts the content of the block with erasure coding (details discussed in §6.2). This is called the “announce” phase.
2. The validators check the validity of the block header, sign the block header with a BLS signature, and send the signature back to the leader.
3. The leader waits for at least $2f+1$ valid signatures from validators (including the leader itself) and aggregates them into a BLS multi-signature. Then the leader broadcasts the aggregated multi-signature along with a bitmap indicating which validators have signed. Together with Step 2, this concludes the “prepare” phase of PBFT.
4. The validators check that the multi-signature has at least $2f+1$ signers, verify the transactions in the block content broadcasted from the leader in Step 1, sign the received message from Step 3, and send it back to the leader.
5. The leader waits for at least $2f+1$ valid signatures (can be different signers from Step 3) from Step 4, aggregates them together into a BLS multi-signature, and creates a bitmap logging all the signers. Finally, the leader commits the new block with all the multi-signatures and bitmaps attached, and broadcasts the new block for all validators to commit. Together with Step 4, this concludes the “commit” phase of PBFT.

The validators of Harmony’s consensus are elected based on Proof-of-Stake. Therefore, the actual protocol differs slightly from the one described above in a sense that a validator with more voting shares has more votes than others, rather than one-signature-one-vote. So instead of waiting for at least $2f+1$ signatures from validators, the leader waits for signatures from the validators who collectively possess at least $2f+1$ voting shares. The details of the proof-of-stake election mechanism will be discussed in §3.3.

3. Sharding

Blockchain sharding as a scalability solution has gained lots of attention since late 2017. Various sharding solutions have been proposed both in industry and academia.

In industry, Zilliqa [12] was the first sharding-based public blockchain that claimed a throughput of 2,800 TPS. Zilliqa uses PoW as identity registration process (i.e. Sybil attack [1] prevention). Zilliqa’s network contains a single directory-service committee and multiple shard committee (i.e. *network sharding)*, each containing hundreds of nodes. Transactions are assigned to different shards and processed separately (i.e. *transaction sharding)*. The resulting blocks from all shards are collected and merged at the directory-service committee. Zilliqa is not a *state sharding* solution because each node has to hold the entire blockchain state to be able to process transactions.

In academia, publications like Omniledger [8] and RapidChain [7] have proposed solutions that feature *state sharding* where each shard holds a subset of the blockchain state. Omniledger employs a multi-party computation scheme called RandHound [25] to generate a secure random number, which is used to randomly assign nodes into shards. Omniledger assumes a *slowly* *adaptive* corruption model where attackers can corrupt a growing portion of the nodes in a shard over time. Under such security model, a single shard can be corrupted eventually. Omniledger prevents the corruption of shards by reshuffling all nodes in the shards at a fixed time interval called *epoch*. RapidChain builds on top of Omniledger and proposes the use of the *Bounded Cuckoo Rule* to reshuffle nodes without interruptions [19].

Harmony draws inspiration from these three previous solutions [7,8,12] and designs a PoS-based full sharding scheme that's linearly scalable and provably secure. Harmony contains a beacon chain and multiple shard chains. The beacon chain serves as the randomness beacon and identity register, while the shard chains store separate blockchain states and process transactions concurrently. Harmony proposes an efficient algorithm for randomness generation by combining Verifiable Random Function (VRF) and Verifiable Delay Function (VDF). Harmony also incorporates PoS in the sharding process which shifts the security consideration of a shard from the minimum number of nodes [7,8,12] to the minimum number of voting shares.

3.1 Distributed Randomness Generation

Background

Various approaches have been proposed to assign nodes into shards such as randomness-based sharding [7,8], location-based sharding [34], and centrally-controlled sharding [35]. Out of all the approaches, randomness-based sharding has been recognized as the most secure solution. In randomness-based sharding, a mutually agreed random number is used to determine the sharding assignment for each node. The random number must have the following properties:

1. Unpredictable: No one should be able to predict the random number before it is generated.
2. Unbiaseable: The process of generating the random number should not be biasable by any participant.
3. Verifiable: The validity of the generated random number should be verifiable by any observer.
4. Scalable: The algorithm of randomness generation should scale to a large number of participants.

Omniledger [8] uses the RandHound [25] protocol, which is a leader-driven distributed randomness generation (DRG) process that involves PVSS (Publicly Verifiable Secret Sharing) and Byzantine Agreement. RandHound is an $O(n * c^2)$ protocol that divides participant nodes into multiple groups of size c. It achieves the first three properties above but is impractically slow to qualify as scalable.

RapidChain [7] takes a simpler approach by letting each participant perform VSS (Verifiable Secret Sharing) [22] and using the combined secret shares as the resulting randomness. Unfortunately, this protocol is not secure because the malicious nodes can send inconsistent shares to different nodes [25]. Besides, RapidChain does not describe how the nodes reach consensus on the multiple possible versions of reconstructed randomness.

In addition, Algorand [18] relies on the VRF-based (Verifiable Random Function) cryptographic sortition to select the group of consensus validators. The Ethereum 2.0 design proposes the use of VDF (Verifiable Delay Function) [20] to delay the revelation of the actual random number so as to prevent last-revealer attack [36]. The VDF is a newly invented cryptographic primitive; it takes an adjustable minimum amount of time to compute and the result can be verified immediately.

Scalable Randomness Generation with VRF and VDF

Harmony's approach combines the strengths of the solutions above. First, Harmony's DRG protocol complexity is $O(n)$, which in practice is at least an order of magnitude faster than RandHound. Second, unlike RapidChain's simple VSS-based approach, ours is unbiasable and verifiable. Third, compared to Ethereum 2.0's solution, our approach uses BFT consensus to provide finality to the random number. Specifically, the protocol includes the following steps:

1. A leader sends an *init* message with the hash of the last block $H(B_{n-1})$ to all the validators.
2. For each validator i, after receiving the *init* message, a VRF is computed to create a random number r_i and a proof p_i: $(r_i, p_i) = VRF(sk_i, H(B_{n-1}), v)$, where sk_i is the secret key of validator i and v is the current view number of consensus. Then, each validator sends back (r_i, p_i) to the leader.
3. The leader waits until it receives at least $f+1$ valid random numbers and combines them with an XOR operation to get the preimage of the final randomness $pRnd$.
4. The leader runs BFT (discussed in §2) among all the validators to reach consensus on the $pRnd$ and commit it in block B_n.
5. After $pRnd$ is committed, the leader starts computing the actual randomness $Rnd = VDF(pRnd, T)$, where T is the VDF difficulty and is set algorithmically such that the randomness can only be computed after k blocks.
6. Once Rnd is computed, the leader initiates a BFT among all validators to agree on the validity of Rnd and finally commit the randomness into the blockchain.

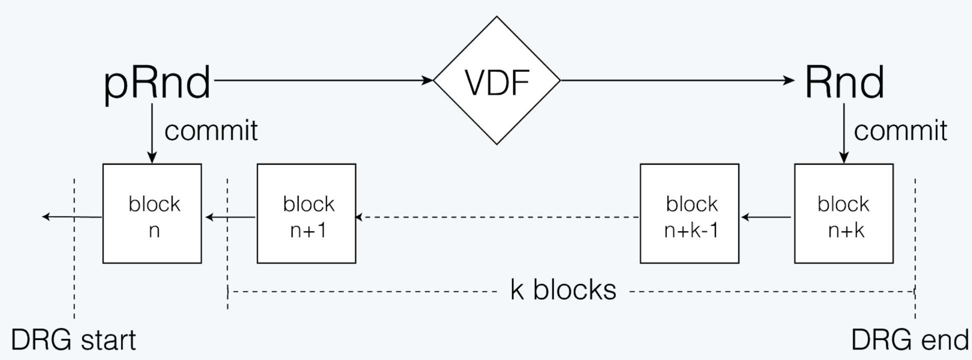

Figure 2. The VDF (Verifiable Delay Function) delays the revelation of the final randomness.

The VDF is used to provably delay the revelation of Rnd and prevent a malicious leader from biasing the randomness by cherry-picking a subset of the VRF random numbers. Because of the

VDF, the leader won't be able to know the actual final randomness before $pRnd$ is committed to the blockchain. By the time Rnd is computed with the VDF, $pRnd$ is already committed in a previous block so the leader cannot manipulate it anymore. Therefore, the best a malicious leader can do is to either blindly commit the randomness $pRnd$, or stall the protocol by not committing $pRnd$. The former is the same as the honest behavior. The latter won't cause much damage as the same timeout mechanism in PBFT [14] will be used to switch the leader and restart the protocol.

We assume, in the long run, the existence of ASICs to compute VDFs, where a few altruistic nodes running an ASIC (Application-Specific Integrated Circuit) will publish the result, and no one could game the system. It is possible, before VDF ASICs are in production, that an attacker with a faster computing device could calculate the result before other honest nodes. Until this happens, the attacker can only know the randomness slightly before the honest nodes. While in principle the attacker could take advantage of this (e.g. withdrawing its fund if the bet on a smart contract was unfavorable to him), this problem can be mitigated on the smart contract layer with a proper delay, such that there should be a waiting period for the randomness to be committed to the protocol before a fund withdrawal is made possible.

3.2 Epochs

In Harmony, the consensus and sharding process is orchestrated by the concept of epochs. An epoch is a predetermined time interval (e.g. 24 hours) during which the sharding structure is fixed and each shard continuously runs consensus with the same set of validators. At the beginning of each epoch, a random number will be generated using the DRG protocol described in §3.1, and the sharding structure will be determined based on that randomness. Validators who want to validate transactions in epoch e need to stake their tokens during epoch $e-1$. The cutoff time for staking is before the randomness preimage $pRnd$ is committed into the blockchain.

3.3 Staking-based Sharding

Validator Registration

Sybil attack [1] prevention is a key security consideration in public blockchains. Bitcoin and Ethereum require the miners to compute a cryptographic puzzle (PoW) before they can propose a block. Similarly, sharding-based blockchains like Zilliqa [12] or Quarkchain [11] also use PoW to prevent Sybil attacks. Harmony adopts a different approach with proof-of-stake (PoS) as the validator registration or Sybil attack prevention mechanism. In order to become a Harmony validator, prospective participants (or stakers) have to stake a certain amount of tokens to be eligible. The number of tokens staked will determine the number of voting shares assigned to the validator. Each voting share corresponds to one vote in the BFT consensus (as discussed in §2).

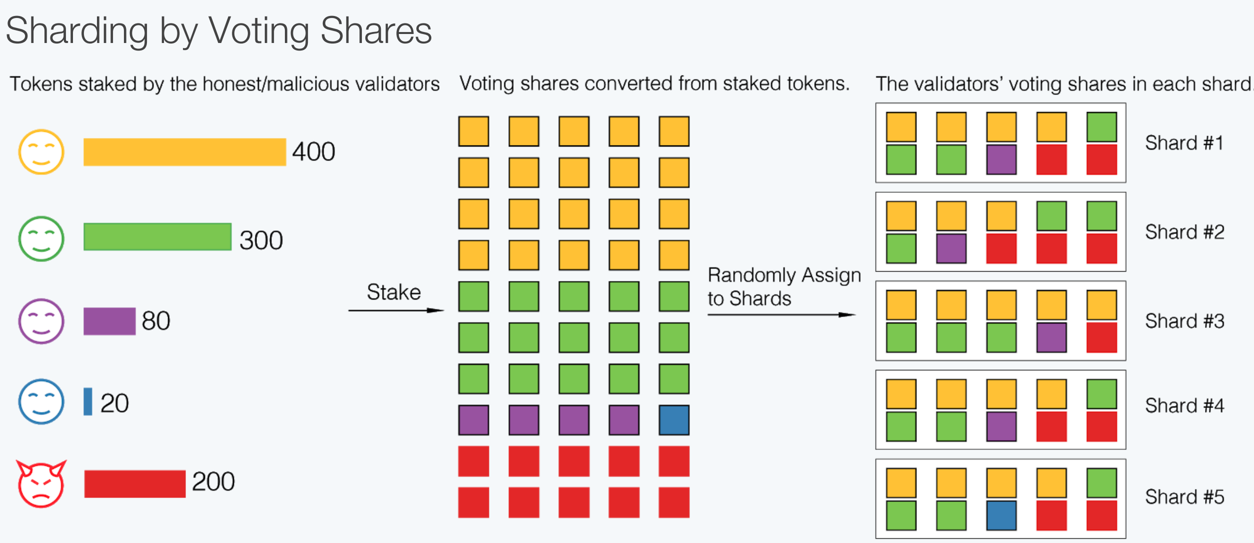

Figure 3. The stakers obtain voting shares proportional to their staked tokens. Voting shares are then randomly assigned to shards. Stakers become validators for the shard(s) where their voting shares are assigned.

A voting share is a virtual ticket that allows a validator to cast one vote in the consensus. Validators can acquire voting shares by staking tokens. The amount of tokens required for a voting share is algorithmically adjusted. At the beginning of each epoch, new validators' voting shares will be randomly assigned to shards. The new validators join the shard(s) where their voting shares get assigned. As discussed in §2, the consensus in a shard is reached by validators who collectively possess at least $2f+1$ voting shares to sign the block.

To guarantee the security of a single shard, the amount of voting shares by malicious validators needs to be kept below $\frac{1}{3}$ of all the voting shares in that shard. This is required due to the nature of BFT consensus. Harmony's adaptive thresholded PoS guarantees the above security requirement by adaptively adjusting the price of a voting share and assigning individual voting shares to shards rather than individual validators.

Our security assumption is that across all the staked tokens, up to $\frac{1}{4}$ of them belong to malicious validators. If we shard by validators (i.e. assign one validator to one shard), in the worst case where a single malicious validator holds $\frac{1}{4}$ of all the staked tokens (or the voting shares), it will easily possess more than $\frac{1}{3}$ voting shares in that shard. The reason is that the stakes at each shard is m times less than the stakes of the whole network, where m is the number of shards. We call this attack scenario a *large-stake attack* (a special type of single-shard takeover attack).

To prevent *large-stake attack*, instead of sharding by validators, we shard by voting shares (i.e. assign one voting share to one shard). Specifically, after the Rnd is revealed at the start of the current epoch, a random permutation (seeded with Rnd) on all the voting shares will be done and the permuted list of voting shares will be divided evenly into m buckets, where m is the number of shards. The voting shares falling in the ith bucket are assigned to shard i, so are the corresponding validators. In practice, a single validator may be assigned to multiple shards if he possesses voting shares assigned to those shards. The shard leader is determined as the validator who possess the first voting share in the bucket.

It's worth noting that validators with larger stakes will have more chance of being selected as the leader. We argue that it's actually a desirable scenario because large stakers have more incentive to follow the protocol due to the fear of their stake being slashed (incentive mechanism is discussed in §7). In addition, they are also more likely to possess more powerful machines with fast and stable network.

Adaptive-Thresholded PoS

The price of a voting share is set algorithmically so that it's small enough that malicious stakers can not concentrate their voting power in a single shard. Specifically, we set the price of a voting share to be P_{vote} tokens:

$$P_{vote} = \frac{TS_{e-1}}{NumShard * \lambda}$$

Here λ is a security parameter, $NumShard$ is the number of shards and TS_{e-1} is the total amount of tokens staked during epoch $e-1$.

Now we prove that when $\lambda > 600$, the chance of a single shard having more than $\frac{1}{3}$ malicious voting shares (i.e. probability of failure) is negligible.

Given the definition of P_{vote}, the total number of voting shares will be $N = \frac{TS_{e-1}}{P_{vote}} = NumShard * \lambda$. Given a trustable randomness source (discussed in §3.1) and the sharding process based on the randomness, the probability distribution of the number of malicious voting shares in each shard can be modeled as a hypergeometric distribution (i.e. random sampling without replacement):

$$P(X = k) = \frac{\binom{K}{k}\binom{N-K}{n-k}}{\binom{N}{n}}$$

Here N is the total number of voting shares, $K = \frac{N}{4}$ is the maximum number of malicious voting shares, $n = \frac{N}{NumShard}$ is the number of voting shares in each shard, and k is the number of malicious voting shares in a shard. The actual failure rate of a shard $P(X \le k)$ follows cumulative hypergeometric distribution $CDF_{hg}(N, K, n, k)$ which, when N is large, degrades to binomial distribution (i.e. random sampling with replacement):

$$P(X \le k) = \sum_{i=0}^{k} \binom{n}{i} p^i (1-p)^{n-i}$$

We can show that when n is large enough, the probability that a shard contains more than $\frac{1}{3}$ tokens held by malicious entities is negligible. In fact, when $n = 600$, the probability that a shard contains less than $\frac{1}{3}$ malicious voting shares is $P(X \leq 200) = 0.999997$, which translates to a shard failure (i.e. consensus cannot be reached) rate of "once in around 1000 years" (given an epoch interval of 24 hours). Therefore, we will set $\lambda = 600$ to guarantee the high security of our shards. (Intuitively, λ governs minimum number of voting shares a single shard should contain. This is functionally similar to the minimum number of nodes in a shard as described in other PoW-based sharding solutions [7,8,12])

This approach is resistant to the fluctuation of the number of validators. We are not setting a lower limit on number of validators in each shard as in other solutions like Zilliqa [12]. Instead, we adopt an adaptive PoS-based model to ensure that the malicious people can never occupy more than $\frac{1}{3}$ of the voting shares in a single shard, thus making it secure.

3.4 Resharding

We've described a secure sharding scheme that prevents malicious validators from overtaking a single shard. Nonetheless, if the sharding structure stays fixed, malicious attackers can still overtake a shard by corrupting the validators in that shard. There are three models of attackers:

1. Static Round-Adaptive: where attackers can only corrupt a subset of nodes at a predetermined stage. Elastico [9] assumes attackers can only corrupt nodes at the beginning of each epoch.
2. Slowly Adaptive: where attackers can corrupt a subset of nodes over time during the epoch [7,8].
3. Fully Adaptive: where attackers can corrupt a subset of nodes instantaneously and at any time [18]

Harmony assumes the slowly adaptive corruption model under which the attacker can corrupt a constant number of nodes and it takes a certain amount of time. Omniledger [8] assumes the same corruption model and it prevents the attack by replacing validators in all shards every epoch. This approach has two major problems. The first is the high cost of bootstrapping at every epoch. The second is the security concern when all nodes are being replaced during the consensus.

Harmony mitigates these problems by adopting the Cuckoo-rule based resharding mechanism [7,19]. After the end of an epoch, the validators who withdrew their stake will be evicted from the network, while those who keep their stakes stay. The new validators who staked during this epoch get new voting shares. These voting shares will be randomly assigned to the shards who have more than the median of the total voting shares. Next, a constant number of the voting shares from all shards will be randomly re-distributed to the other half of the shards who have less than the median of total voting shares. It's proven in [7] that this resharding scheme can keep the voting shares in all shards balanced while fulfilling the security requirement.

3.5 Fast State Synchronization

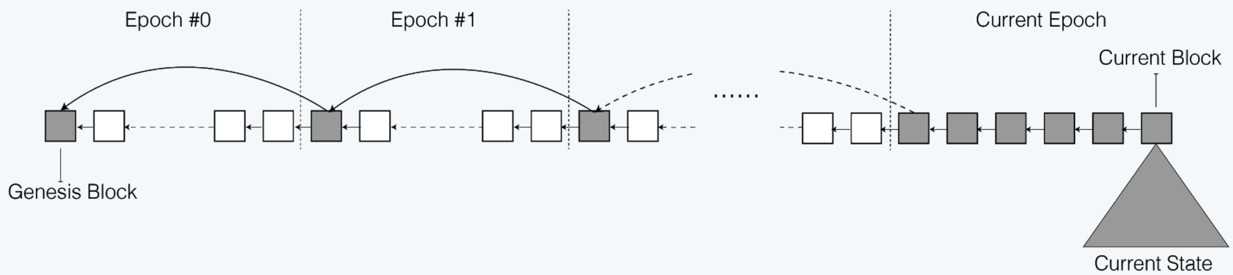

Figure 4. The first block of an epoch contains a hash link to the first block of last epoch. This allows fast state synchronization of new nodes where they can rely only on the blocks in grey to quickly verify the current state.

When validators join a new shard, they will need to quickly synchronize to the current state of the shard in order to validate new transactions. The traditional procedure of downloading the blockchain history and reconstructing the current state is too slow for resharding to be possible (it takes days to fully synchronize the Ethereum blockchain history). Fortunately, the current state is orders of magnitude smaller than the whole blockchain history. Downloading the current state within the time window of an epoch is feasible compared to downloading the whole history.

In Harmony, new validators joining a shard first download the current state trie of that shard so they can start validating transactions quickly. To ensure the current state downloaded is valid, the new node needs to do proper verification. Instead of downloading the whole blockchain history and replaying all the transactions to validate the current state, the new node downloads historical block headers and validates the headers by checking their signatures. As long as there is a cryptographic trace (e.g. hash pointers and signatures) from the current state back to the genesis block, the state is valid. Nonetheless, signature verification is not computationally free and it takes a significant amount of time to verify all the signatures starting from the genesis block. To mitigate this problem, the first block of each epoch will include an additional hash pointer to the first block of the last epoch. This way, the new node can jump across the blocks within an epoch when tracing hash pointers to genesis block. This will significantly speed up the verification of the current blockchain state.

To further optimize the state synchronization process, we will make the blockchain state itself as small as possible. One observation from the Ethereum blockchain state is that a lot of accounts are empty and wasting the precious space of blockchain state. In Ethereum, the empty accounts with a specific nonce cannot be deleted because of potential replay attacks where old transactions are re-submitted on the deleted account [32]. Harmony will adopt a different model of avoiding replay attacks by letting the transactions specify the hash of the current block: a transaction is only valid before a certain number (e.g. 100) of blocks following the block of the specified hash. This way, the old accounts can be safely deleted and the blockchain state can be kept slim.

4. Shard Chain and Beacon Chain

4.1 Shard Chain

A shard chain is a blockchain that processes and validates its own transactions and stores its own state. A shard only processes transactions that is relevant to itself. Although a shard chain is relatively independent, it will communicate with other shard chains through cross-shard communication.

Cross-shard Communication

Cross-shard communication is a key component of any sharding-based blockchain. Cross-shard capability breaks the barrier between shards and extends the utility of a single shard beyond itself. Overall, there are three categories of cross-shard communication:

1. Main-chain-driven: Projects like Zilliqa [12] rely on the main chain to achieve transactions across shards.
2. Client-driven: Omniledger [8] proposed a client-driven cross-shard transaction mechanism where the messages between shards are collected and sent to shards by clients. This adds an extra burden to the client that is not desirable for an adhoc light client.
3. Shard-driven: RapidChain [7] proposed that the messages between shards are directly sent by the nodes in the shard without external help.

Harmony adopts the shard-driven approach for its simplicity and the absence of burden on clients. We believe the benefits of shard-driven communication outweighs its drawbacks. The cost on the overall network for shard-driven communication can be considerable because every cross-shard message is a network-level broadcast, which incurs a $O(N)$ network cost. To solve this problem, Harmony uses the Kademlia routing protocol to reduce the communication complexity to $O(log(N))$. In addition, the data being communicated will be encoded with erasure code to ensure the robustness of cross-shard communication. The details will be discussed in §6.

4.2 Beacon Chain

The Harmony beacon chain is a special blockchain that serves additional purposes compared to the shard chains. In effect, the beacon chain is also a shard. Besides processing transactions, like other shard chains do, the beacon chain is in charge of two additional key functionalities: generating the random number (discussed in §3.1) and accepting stakes, which means that the beacon chain is the chain where stakers deposit their tokens to become validators.

The validators for the beacon chain are determined similarly as the other shard chains are. During the sharding assignment, the voting shares are randomly divided into $NumShard + b$ buckets, where the extra b buckets are for the beacon chain.

Hash Link from Shard Chain

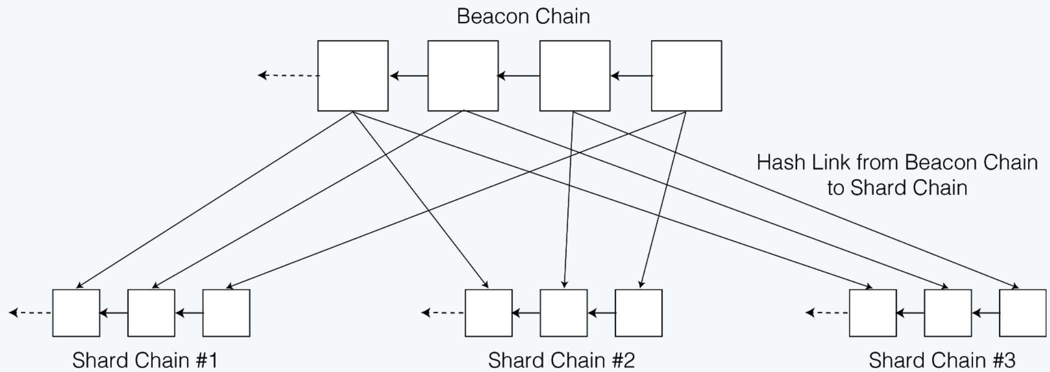

Figure 5. Hash link from beacon chain block to shard chain block.

The beacon chain helps strengthen the security and consistency of the shard chains' states by including the block header from each shard chain. Specifically, after a new block is committed to a shard chain, its block header will be sent (via Kademlia-based inter-shard communication) to the beacon chain. The beacon chain checks the validity of the block header by:

1. The hash of its previous block, which must have already been committed in the beacon chain;
2. The signers of the block's multi-signature, which must be the correct validators for that shard.

The committed block headers at the beacon chain will then be broadcasted to the whole network. Each shard will keep a chain of valid block headers for all other shards, which will be used to check the validity of transactions from other shards (i.e. simple payment verification). Adding the shard chains' block headers into the beacon chain serves two main purposes:

1. Increases the difficulty of attacking a single shard.
   Attackers have to corrupt both the shard chain and beacon chain in order to convince others that an alternative block in the shard chain is valid.
2. Reduce the network cost of broadcasting the block headers among shards.
   There will be a $O(N^2)$ network communication if we let each shard broadcast its headers separately. With the beacon chain as a central relay, the complexity is reduced to $O(N)$.

5. Blockchain State Sharding

Unlike other state-sharding blockchains [7,8] that adopted UTXO (Unspent Transaction Output) data model, Harmony's state sharding is applied on account-based data model. Each shard chain contains its own account state, and all the tokens in existence are spread among all the shard.

We treat the user account and the smart contract account differently in sharding. An user account can have multiple balances at different shards (e.g. 100 tokens at Shard A and 50 tokens at Shard B). A user account can move its balance between shards by issuing a cross-shard transaction. A smart contract account is limited to the specific shard where the contract was created. However, for a decentralized application that requires more throughput than a single shard can handle, the Dapp (Decentralized Application) developer can instantiate multiple instances of the same smart contract in different shards and let each instance handle a subset of the incoming traffic. Note that the different instances of the same smart contract do not share the same state, but they can talk to each other via cross-shard communication.

6. Networking

Previous research [33] has pointed out that network capacity is one of the major bottlenecks for blockchain systems. In order to increase performance, Harmony focuses on improving the efficiency of network utilization. Harmony also proposes a number of improvements to deal with real-world networking scenarios.

6.1 Kademlia-based Routing

Inspired by RapidChain [7], we will adopt Kademlia [37] as the routing mechanism for cross-shard messages. Each node in the Harmony's network maintains a routing table that contains nodes from different shards. The distance between shards is defined as the XOR distance of the shard IDs. When a message from shard A needs to be sent to shard B, the nodes in shard A will look at the routing table and send the message to the nodes with the closest shard ID. With Kademlia-based routing, a message only travels across $O(logN)$ nodes before it reaches the destination shard. Compared to normal gossip broadcasting, which requires a $O(N)$ network complexity, the Kademlia routing mechanism can significantly reduce the overall network load in a sharded blockchain.

6.2 Efficient Broadcasting with Erasure Code

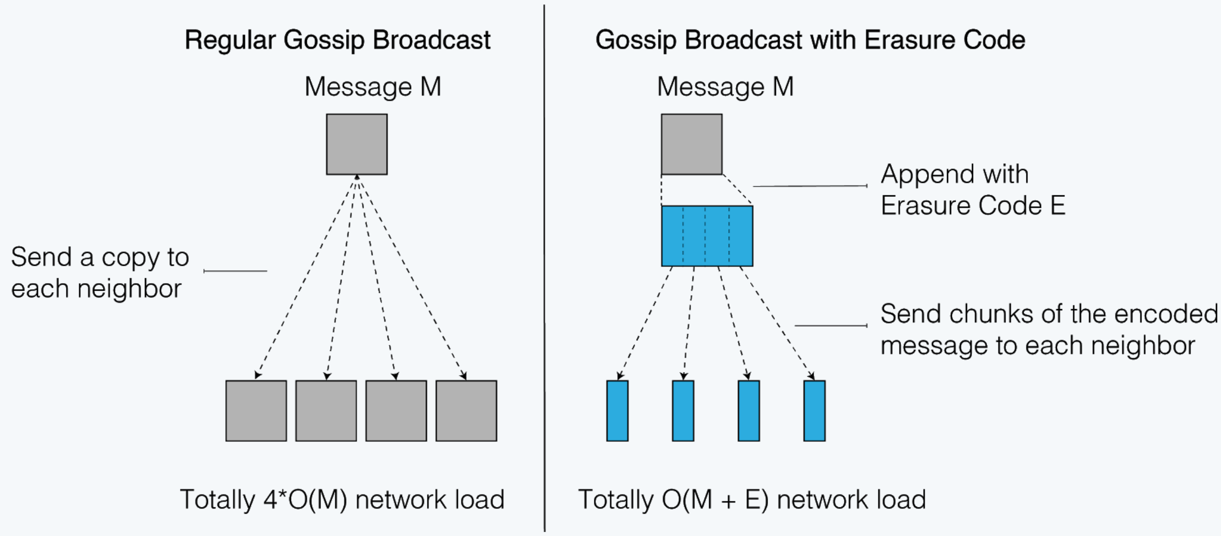

Figure 6. Comparison between normal gossip broadcast with gossip broadcast with erasure code.

Broadcast is a frequent network action in any blockchain system built on P2P (Peer-to-Peer) network overlay. Specifically in our consensus protocol, there are three scenarios where broadcasting is needed:

1. A newly proposed block needs to be broadcasted by the leader to all validators.
2. A newly generated master chain block needs to be broadcasted to the whole network.
3. The cross shard communication requires the broadcast of a message between shards.

In a normal P2P broadcasting, the original sender needs to send a copy of the message to each of its neighbors. This will incur $O(d * M)$ network load on the sender, where d is the average number of neighbors of the sender and M is the message size. Instead, in Harmony, a sender first encodes the message with erasure code and then send chunks of the encoded message to each neighbor. This reduces the load on the sender to $O(M + e)$ where e is the size of erasure code and it is usually smaller than the size of the original message M. Therefore, Harmony's network broadcasting mechanism significantly lowers the network load of the broadcast sender. In addition, Harmony proposes to improve IDA's robustness by replacing the original Reed-Solomon erasure code with RaptorQ fountain code so that the broadcaster can always send more erasure codes to further ensure the data is eventually received.

6.3 FEC-based Unicast

Traditional reliable transports such as TCP [42] relies upon retransmission and ACK-based signaling in order to deal with lost packets. This is known to introduce latency spikes proportional to the round-trip time between the sender and receiver. Also, window-based congestion control—such as Reno, NewReno, and CUBIC in use by most TCP implementations—are all additive increase/multiplicative decrease (AIMD) algorithms, whose bandwidth is known to be severely impacted by transient packet losses.

Harmony uses the RaptorQ fountain code to combat these two problems. Each message is encoded into symbols, and symbols are sent over the wire until the receiver acknowledges successful decoding of the message using the symbols that it received. Unlike using fixed-rate codes such as Reed–Solomon where the transmission fails once the symbols have been exhausted, fountain code enables infinite, just-in-time generation and use of encoding symbols.

6.4 Support for Home Nodes

P2P nodes on a typical residential network pose a major, distinctively unique problem: They cannot be reached from the outside unless mediated by their residential internet router, which employs a technique called network address translation (NAT). Support for inbound traffic by these routers vary, and different approaches have been developed to work around different types of routers. In particular, routers implementing symmetric NAT cannot easily be worked around unless explicitly configured to support other hole-punching mechanisms such as Internet Gateway Device Protocol (IGDP).

Harmony's P2P layer tries to detect the NAT mechanism behind which a node operates and employs the right workaround mechanism, such as STUN, TURN, IGDP, etc. In particular, Harmony implements the overall detection and mitigation protocol named ICE (Interactive Connectivity Establishment).

6.5 Support for Locator Mobility

Nodes may change their IP addresses, with some type of nodes more so than others. One such example is a laptop, which may frequently hop between different Wi-Fi networks, with its IP address changing each time. When an IP address of a node changes, all existing transport connections that use the IP address as a local or remote endpoint are interrupted, and applications directly using such transport connections need to re-establish connections using the new IP address in order to continue. Such a connection handover is hard to implement correctly with minimal application-layer service interruption. Also, handling connection handover often complicates application-layer protocols (such as base consensus protocols).

Harmony's network layer, in order to solve this problem, introduces a clean separation between node identity (cryptographic key pair possessed by the node) and node locators (network/transport-layer locator where the node can be reached) using the industry-standard Host Identity Protocol Version 2 (HIPv2). HIPv2 lets locators of a node change over time while keeping the node identity, by providing mechanisms for locator discovery, node-to-node security association, and tunneling of upper-layer traffic associated with local/remote node identity as endpoints.

7. Incentive Model

7.1 Consensus Rewards

After the successful commitment of a block, a protocol-defined number of new tokens will be rewarded to all validators who signed the block in proportion to their voting shares. The transactions fees are rewarded to validators similarly.

7.2 Stake Slashing

For any misbehaviors detected by the network, a certain amount of staked tokens will be slashed. For example, if a leader failed to finish the consensus process and triggered the leader change process, P_{vote} staked tokens will be slashed. If validators are proven to sign a dishonest block, all of their stake under the same shard will be slashed. This severe punishment is meant to strongly discourage any dishonest behavior and make the network as secure as possible. A proof of misbehavior can be two signed blocks that conflict with each other. Any validator can submit a transaction to prove the misbehavior of other validator and if verified, the slashed token will be rewarded to the prover(s).

7.3 Stake withdrawal

Long-range Attacks

Proof-of-stake blockchains, unlike proof-of-work blockchains, tend to suffer from *long-range attacks*. These are attacks that leverage the fact that proofs are based on signatures rather than on resource-intensive tasks. In a long-range attack, the private keys of honest validators are stolen long after they have been used, and the attacker is able to create a forked blockchain by signing fake blocks with those keys. When this happens, new validators joining the network have no way to distinguish between the original, legitimate chain and the attacker's simulated chain.

Long-range attacks happen in the following two scenarios. Private key can be compromised either by a lack of security on validators, or more commonly, by the fact, after a validator withdraw their token, he could financially benefits if an attacker which would be looking to buy its private key. Also, by design each set of validators is trusted to approve the block of transactions that also determines the next set of validators. After enough private key (i.e. those that collectively hold more than $\frac{2}{3}$ voting shares in a shard) has been compromised, an attacker has total control on who the subsequent validators is.

Long-range Defense: Resonant Quorums

Proof-of-work blockchain protects against the above attacks by giving honest validators an objective method of fork choice. In a proof-of-work blockchain, the fork choice to select the canonical chain is the accumulated amount of work done in terms of hashes computed.

In a proof-of-stake blockchain, the only objective measure that can be used to select between forks is the total weighting of signatures used to approve each block. If we use these weighted signatures to compare two different blocks, we come to the following equation to determine when a chain may be forked:

Safety = "Block approval key weight" - "Compromised key weight"

The "Block approval key weight" means the voting power of the keys that signed on the block. If, by stake weight, more private keys are compromised than were used to approve of a block, then the block can be forked. Until then, validators will always prefer the original, legitimate version of the block.

Harmony maximizes the safety of each block in its proof-of-stake blockchain by maximizing this equation. It is infeasible to disincentivize leaking private keys in the long term. Harmony instead incentivizes validators to maximize the approval weight of each block after a quorum has been achieved. This is done by requiring validators to sign each quorum-approved block before allowing those validators to withdraw their stake. These new additional signatures only need to exist within the blockchain, and they do not need to be generated at consensus time for each block. Because of this, the new signatures can be added to subsequent blocks when validators decide to withdraw their stake, and so they may freely improve the safety of the chain without impacting its liveness.

8. Future Research

8.1 Fraud Proofs

The capability of proving the misbehavior of validators is important for a light client to trust the block data they received. In the case of cross-shard communication, each shard is a light client of other shards. Ensuring that messages sent between shards are trustable is crucial for inter-shard data consistency. We are actively researching the topic of data availability [29] and fraud proofs [2] to securitize our protocol.

8.2 Stateless Validators

In a high throughput blockchain, the size of the blockchain data will grow faster than existing chains, which is a major problem for new validators to sync up quickly. This makes the resharding process problematic because if new validators can't sync up in time, then the quorum of validators may not be reached for a new block to be approved, and even if the quorum is met, the security of the protocol would be reduced. State block pruning is one mitigation to the problem, but it's not optimal since the state itself can grow large. We are actively looking into enabling stateless client [30,31] where validators doesn't have to sync up the full state to validate transactions.

References

[1] J.R. Douceur, The Sybil attack, in: 1st International Workshop on Peer-to-Peer Systems (IPTPS 02), 2002.

[2] Al-Bassam, M., Sonnino, A., & Buterin, V. (2018). Fraud Proofs: Maximising Light Client Security and Scaling Blockchains with Dishonest Majorities. CoRR, abs/1809.09044.

[3] Vasin, P. (2014) Blackcoin's Proof-of-Stake Protocol v2, https://blackcoin.co/blackcoin-pos-protocolv2-whitepaper.pdf

[4] A. Kiayias, I. Konstantinou, A. Russell, B. David, and R. Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. Cryptology ePrint Archive, Report 2016/889, 2016. http://eprint.iacr.org/.

[5] P. Daian, R. Pass and E. Shi, Snow White: Robustly reconfigurable consensus and applications to provably secure proofs of stake, Cryptology ePrint Archive, Report 2016/919, 2017.

[6] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. https://eprint.iacr.org/2017/913.pdf.

[7] M. Zamani, M. Movahedi, and M. Raykova, "RapidChain: A Fast Blockchain Protocol via Full Sharding." Cryptology ePrint Archive, Report 2018/460, 2018. https://eprint.iacr.org/2018/460.

[8] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in 2018 IEEE Symposium on Security and Privacy (SP), pp. 19–34, 2018.

[9] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16, pages 17–30, New York, NY, USA, 2016. ACM.

[10] George Danezis and Sarah Meiklejohn. Centrally banked cryptocurrencies. In 23rd Annual Network and Distributed System Security Symposium, NDSS, 2016.

[11] The QuarkChain Team. Cross Shard Transaction. https://github.com/QuarkChain/pyquarkchain/wiki/Cross-Shard-Transaction

[12] The Zilliqa Team. The zilliqa technical whitepaper. https://docs.zilliqa.com/whitepaper.pdf, August 2017.

[13] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. Available at https://bitcoin.org/ bitcoin.pdf.

[14] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance. In Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI '99), New Orleans, Louisiana, February 1999.

[15] E. Kokoris-Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. In Proceedings of the 25th USENIX Conference on Security Symposium, 2016.

[16] Drijvers, M., Edalatnejad, K., Ford, B., & Neven, G. (2018). Okamoto Beats Schnorr: On the Provable Security of Multi-Signatures. IACR Cryptology ePrint Archive, 2018, 417.
[17] B. Alangot M. Suresh A. S Raj R. K Pathinarupothi K. Achuthan "Reliable collective cosigning to scale blockchain with strong consistency" Proceedings of the Network and Distributed System Security Symposium (DISS'18) 2018.
[18] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. Cryptology ePrint Archive, Report 2017/454, 2017.
[19] Baruch Awerbuch and Christian Scheideler. Towards a scalable and robust DHT. In Proceedings of the Eighteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '06, pages 318–327, New York, NY, USA, 2006. ACM.
[20] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In CRYPTO 2018, 2018.
[21] D. Boneh, B. B¨unz, and B. Fisch. A survey of two verifiable delay functions. Cryptology ePrint Archive, Report 2018/712, 2018. https://eprint.iacr.org/2018/712.
[22] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In Proceedings of the 28th Annual Symposium on Foundations of Computer Science, SFCS '87, pages 427–438, Washington, DC, USA, 1987. IEEE Computer Society.
[23] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford. Keeping Authorities "Honest or Bust" with Decentralized Witness Cosigning. In 37th IEEE Symposium on Security and Privacy, May 2016.
[24] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. CoRR, abs/1710.09437, 2017.
[25] E. Syta, P. Jovanovic, E. Kokoris-Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford. Scalable Bias-Resistant Distributed Randomness. In 38th IEEE Symposium on Security and Privacy, May 2017.
[26] T. Hanke, M. Movahedi, and D. Williams. Dfinity technology overview series consensus system, January 2018.
[27] The Ethereum Foundation. Ethereum Whitepaper. https://github.com/ethereum/wiki/wiki/White-Paper.
[28] D. Boneh, B. Lynn, and H. Shacham. Short Signatures from the Weil Pairing. In Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology, ASIACRYPT '01, pages 514–532, London, UK, UK, 2001. Springer-Verlag. https://www.iacr.org/archive/asiacrypt2001/22480516.pdf
[29] The Ethereum Team. A note on data availability and erasure coding. https://github.com/ethereum/research/wiki/A-note-on-data-availability-and-erasure-coding
[30] A. Chepurnoy, C. Papamanthou, Y. Zhang. Edrax: A Cryptocurrency with Stateless Transaction Validation. Cryptology ePrint Archive, Report 2018/968.
[31] V. Buterin. The Stateless Client Concept. https://ethresear.ch/t/the-stateless-client-concept/172
[32] Derek Leung, Adam Suhl, Yossi Gilad, and Nickolai Zeldovich. Vault: Fast bootstrapping for cryptocurrencies. Cryptology ePrint Archive, Report 2018/269, 2018.
[33] Ethereum Wiki. On Sharding Blockchain. https://github.com/ethereum/wiki/wiki/Sharding-FAQs

[34] M. F. Nowlan, J. Faleiro, and B. Ford. Crux: Locality-preserving distributed systems. CoRR, abs/1405.0637, 2014.
[35] George Danezis and Sarah Meiklejohn. Centrally banked cryptocurrencies. In 23nd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016. The Internet Society, 2016.
[36] Paul Dworzanski. A note on committee random number generation, commit-reveal, and last-revealer attacks. http://paul.oemm.org/commit_reveal_subcommittees.pdf.
[37] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In Revised Papers from the First International Workshop on Peer-to-Peer Systems, IPTPS '01, pages 53–65, London, UK, UK, 2002. Springer-Verlag.
[38] David K. Gifford, Doctoral Dissertation, Information Storage in a Decentralized Computer System.
[39] Prince Mahajan, Lorenzo Alvisi, and Mike Dahlin Consistency, Availability, and Convergence , Technical Report (UTCS TR-11-22) http://www.cs.cornell.edu/lorenzo/papers/cac-tr.pdf
[40] Wyatt Lloyd et. al, Don't Settle for Eventual: Scalable Causal Consistency for Wide-Area Storage with COPS, Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP'11). https://www.cs.cmu.edu/~dga/papers/cops-sosp2011.pdf
[41] Peter Bailis†, Ali Ghodsi, Joseph M. Hellerstein, Ion Stoica, Bolt-on Causal Consistency, [SIGMOD'13].
[42] Postel, J. (1981). Transmission control protocol specification. *RFC 793.*
[43] Luby, M., Shokrollahi, A., Watson, M., Stockhammer, T., & Minder, L. (2011). RaptorQ forward error correction scheme for object delivery (No. RFC 6330).