The 0Chain Consensus Protocol

Jonathan Katz[1]*, Thomas Austin[2], Siva Dirisala[3], and Saswata Basu[3]

[1] Dept. of Computer Science, University of Maryland.
[2] 0Chain LLC and San Jose State University.
[3] 0Chain LLC.

Abstract. We describe the 0Chain blockchain ecosystem, including a new consensus protocol offering fast finality. We provide proofs of security for the protocol, along with experiment results validating its efficiency under realistic network conditions.

1 Introduction

Since the advent of Bitcoin [Nak09], the blockchain has revolutionized the world of cryptocurrencies and distributed computation. Ethereum [Woo14] further developed this promise by integrating Turing-complete smart contracts into the blockchain for building distributed applications (dApps).

Despite the promise of blockchain protocols, they have been held back by their slow consensus times. For example, in Bitcoin a transaction is not considered finalized until it is six blocks deep in the chain, a process which takes roughly one hour. Newer protocols have attempted to address this limitation by introducing consensus algorithms with faster finalization times.

One such protocol, Dfinity [HMW18], uses a randomness beacon (implemented via a *verifiable random function*, or VRF) for ranking different proposed blocks. The designers also introduce the concept of *notaries* who sign the highest-ranked block in each round. The authors describe notarization as "optimistic consensus"; in most rounds, only one block will be notarized, and in that case the unique notarized block will be finalized soon thereafter. Importantly, only notarized blocks will be accepted as part of a chain by miners; this prevents both selfish mining [ES18] and the "nothing-at-stake" problem [Poe15].

0Chain offers a "fast, flexible, free" platform for dApp development through a proof-of-stake consensus protocol that extends previous work in several ways. First, the 0chain protocol assigns various parties in the system specialized roles: at any given time, a subset of the clients (referred to as the "active set") serve as *miners* running the consensus protocol; in turn, a subset of the miners act as *generators* proposing new transactions. *Sharders* store the blockchain history and respond to queries about that history; and *blobbers* store data needed for dApps. This design allows for more specialized machines to be used for each of these roles; by reducing the number of parties running the consensus protocol at any point in time, it also reduces network latency thus improving finalization

* Work done as part of a consultancy agreement with 0Chain LLC.

time. 0Chain also introduces a *squared staking* approach for Sybil resistance, by which miners and sharders are probabilistically chosen based on the *square* of the number of tokens they have staked; this design incentivizes miners and sharders to stake coins using a single account, and thus risk greater penalties should they fail to run the protocol correctly. Finally, the 0Chain consensus protocol makes very mild assumptions about the network latency. This, in particular, allows for faster finalization time because nodes do not need to wait until all messages in a given round have been delivered, but can instead progress shortly after they receive their messages.

In this document we describe the 0Chain ecosystem, focusing primarily on the underlying consensus mechanism.

2 0Chain Entities

There are several roles within the 0Chain ecosystem. A *client* is anyone who possesses 0Chain tokens. At any point in time, some subset of the clients constitute the *active set* and we refer to those clients as *miners*. (The active set is updated on a regular basis, and the mechanism for doing so is detailed in Section 4.) All miners serve as notaries and also implement the randomness beacon. At any given point in time, some subset of the miners also serve as *generators* that are responsible for extending the blockchain by proposing new transactions. By only having a subset of the clients act as miners, and a subset of miners act as generators, we reduce network traffic and decrease the time for messages to flood the network.

The 0Chain ecosystem also includes *sharders* who are responsible for storing older blocks in the blockchain, and *blobbers* who store arbitrary data committed to the blockchain; the set of active sharders is also updated regularly. This division of responsibilities is one of the cornerstones of the 0Chain architecture. Assigning specific tasks to specific entities allows for greater specialization of machines and load-balancing of different tasks. For example, blobbers must be good at storing large amounts of data, but do not need the same computational power a sharder needs. We discuss sharders and blobbers further in Section 5; note, however, that they are not directly relevant to the analysis of the consensus protocol itself.

3 Mining Algorithm

0Chain's mining process follows in the footsteps of [HMW18] to rapidly produce and finalize blocks. Within a given *epoch* (namely, a period of time during which the active set does not change), the blockchain is extended in a sequence of rounds where in each round r the mining process works as follows:

1. The (decentralized) randomness beacon produces a random value ξ_r. This is done exactly as in [HMW18], so we omit further details.

2. The value ξ_r is used to determine which miners will be generators in the current round. Each miner is chosen to be a generator with the same probability (i.e., independent of their stake). The random value is also used to determine a ranking among the generators (as in [HMW18]).
3. Each generator extends its current chain of length $r-1$ by adding a new signed block containing all pending transactions of which it is aware. Each generator sends the updated chain (which includes its new block) to all other miners.
4. After waiting a fixed amount of time (denoted $\mathsf{BlockTime}$), each miner—acting as a notary—considers all the valid, length-r chains it has received. (A chain is valid if it consists of blocks chained back to the genesis block in the usual way, where for $i < r$ the ith block is notarized for round i.) It then ranks these chains by the generator of the rth block. For all highest-ranked chains with rth block B_r, it sends a signed *verification ticket* for B_r (and the corresponding chain) to all miners. Miners continue doing this until there is a notarized block for the current round. (A miner continues issuing verification tickets if new blocks with the same or higher priority arrive after it has already issued a verification ticket for another block, but before the next round has begun.)
5. As in [HMW18], when there are sufficiently many verification tickets for a block, that block is considered *notarized.* We currently consider a block to be notarized only if it has been verified by a strict majority of miners, but it is also possible to require a strict majority of the total stake of the active set.

   Once a miner sees the first notarized block for the current round, it sends the notarization (i.e., the collection of verification tickets[4] for that block) to all miners; sends the notarized block[5] to all sharders; and proceeds to the next round.
6. Our notion of consensus is similar to the one used in Bitcoin. Specifically, in round r each miner and sharder takes their current chain to be any valid, notarized chain of length $r-1$ (ties can be broken arbitrarily). This means there may be temporary disagreement, however we show that miners/sharders disagree on blocks at depth $r-k$ with probability exponentially decreasing in k.

We now prove that the protocol above satisfies safety as long as the honest miners are a strict majority of the active set, and liveness as long as there is at least[6] one honest generator. We assume that all messages are eventually delivered, but make no assumptions about how long such delivery takes.

[4] Note that the block itself need not be sent, since in normal operation all other miners will already have it. If another miner receives verification tickets for a block it does not currently have, it can request that block from other miners.

[5] In fact, it sends only a hash of the notarized block to the sharders; each sharder can then request any block it does not already have from a single miner.

[6] For the purposes of analyzing the protocol in the present section we assume at least one honest generator. In our full protocol, however, we include a mechanism to update the set of generators if no new blocks are added after some specified time.

Theorem 1. *For any r, every honest miner eventually enters round r.*

Proof. An honest miner enters round r only once it has received a notarized block for round $r-1$, which it then forwards to all other miners. So once one honest miner enters round r, all other honest miners will eventually enter round r.

Consider now some honest miner in round $r-1$. As just argued, all honest miners eventually enter round $r-1$, and so the value ξ_r from the randomness beacon is eventually generated and received by all honest miners. Since there is at least one honest generator, there will be at least one block proposed for round $r-1$; moreover, that block will eventually reach all notaries and so at least one round-$(r-1)$ block will be notarized. The notarization is eventually received by the honest miner, who then enters round r.

We say that a round r is *normal*, denoted Normal_r, if only one block is ever notarized for round r. Note that under our stated assumptions (when the adversary can act arbitrarily) this means there is a round-r block B_r such that each honest miner has *only* issued a verification ticket for B_r in round r by the time they enter round $r+1$. (This suffices since honest miners only issue verification tickets for blocks belonging to the current round.)

We say round r is *finalized* if there is a round-r block B_r such that the chain of any honest miner in any round $r' \geq r$ includes B_r. In other words, there is permanent agreement among all honest miners on the block at position r (though the honest miners may not be aware of the fact).

Lemma 1. *If Normal_r occurs then each of rounds $1,\ldots,r$ is finalized.*

Proof. Let B_r be the unique block notarized in round r. The theorem follows from the fact that the chains held by any honest miners at any round $r' \geq r$ must include B_r.

Consider the time when the first honest miner begins some round, and let p_{normal} denote the minimum probability, taken over the random value ξ produced by the randomness beacon and the random latencies[7] of the various links in the network, that the given round is normal (where the minimum is taken over all possible configurations of the honest miners and all possible actions of the attacker). We have:

Theorem 2. *Assume an honest miner is in round r. Then the probability that round $r-k$ is not finalized is at most $(1-p_{\mathsf{normal}})^k$.*

Proof. Consider the honest miner at the beginning of round r who holds a chain of length $r-1$. Round $r-k$ is finalized if any of the k rounds $r-k,\ldots,r-1$ is normal. This implies the theorem.

[7] For the purposes of the present discussion, we treat the distribution over the latency of each link in the network as being fixed for the lifetime of the protocol.

Of course, the theorem is only interesting if $p_{\mathsf{normal}} > 0$. But note that Normal_r occurs, at least, when (1) the generator with highest priority is honest and (2) the message from that generator reaches each honest miner before that miner sends out a verification ticket for round r. In general, p_{normal} depends on the distributions over the latencies of the various links in the network, as well as on the value of $\mathsf{BlockTime}$. Under certain conditioned, it may be possible to rigorously bound p_{normal}; for example, this is possible if one is willing to assume an absolute upper bound Δ on the transmission times between all pairs of miners, as long as $\mathsf{BlockTime}$ large enough relative to Δ. In other cases, bounds on p_{normal} may be obtained via simulation. We include in Appendix A some preliminary experimental results that can be used to bound p_{normal}.

4 Choosing the Active Set

The active set is updated in a series of epochs. A new epoch is begun every predetermined number of rounds, but initiation of a new epoch can also be triggered earlier by the current active set if some miners fall below an expected level of service.

In order to be considered for membership in the next active set, a client must write a transaction to the blockchain staking some number of tokens that they hold. (*Staking* a token means locking it so it cannot be spent; staking also allows punishment if the client staking a token later misbehaves. We refer to [Aus18] for more details.) The staked tokens of any clients who are not selected are immediately unstaked (i.e., become available for that client to spend again). If a client is selected for the active set, its staked tokens remain staked until it leaves the active set in some future epoch.

We wish to incentivize clients to stake tokens under one identity rather than under many separate identities. Therefore, we set the probability of being selected for the active set to be proportional to the *square* of a client's staked tokens.[8] So, for example, if Alice stakes 2 tokens and Bob stakes 1 token, and if only one client is chosen, then the probability for Alice to be chosen is $2^2/(2^2 + 1^2) = 4/5$ and the probability for Bob to be chosen is $1/5$. (In contrast, if probabilities were directly proportional to stake then the probability of Alice being chosen would be $2/3$ and the probability of Bob being chosen is $1/3$.) Note that this encourages Alice to stake her tokens under one name; if she staked 1 token as Alice and another as Ann then the probability of either Alice or Ann being chosen would be $2 * 1^2/(1^2 + 1^2 + 1^2) = 2/3 < 4/5$.

The process for choosing a new active set for the next epoch in some round r is as follows:

1. The value ξ_r from the randomness beacon is used to select 10% of the current miners to be removed from the active set.[9] It is also possible to instead

[8] Any superlinear function could be used.

[9] If there are not enough potential miners to replace the 10% of miners being removed, the fraction of removed miners can be changed.

remove the 10% worst performers in the epoch. We remark that this latter approach potentially allows an attacker to attempt to increase its representation in the active set by always trying to ensure that it is not in the bottom 10% of performers. Another way of looking at this, however, is that the attacker is thereby incentivized to perform well (at least part of the time)!

2. Replacements for the clients who have been removed are selected from among clients who have staked tokens to be selected for the next epoch. As discussed above, each client's chance of being selected is proportional to the square of their total staked tokens.
3. The miners in the new active set have some number of rounds to sync up with the blockchain before they begin mining. The mechanism for doing so will be described in a subsequent document.

5 Sharders and Blobbers

Sharders. Due to the rapid rate at which blocks are produced, it quickly becomes infeasible for miners to store the entire blockchain. Instead, that role is offloaded to another group of entities, called sharders. A given sharder does not store all blocks, either; instead, each sharder stores only a subset of the blocks. The blocks that any given sharder is responsible for are determined by the sharder's ID, using an approach similar to *consistent hashing* [KLL$^+$97]. Specifically, in order to determine which sharders are responsible for storing a given block, a function of each sharder's ID and the block hash is used to generate a value; those values are used to rank all the active sharders, and the top n sharders store that block. In the case of ties, additional sharders may store the block.

Sharders are chosen to be active the same way miners are chosen to be part of the active set, and on the same schedule. However, in each epoch a smaller percentage of sharders is replaced. Note that when a sharder becomes inactive, the only blocks that need to be reallocated are those that were stored by the sharder who is leaving. When a sharder joins, they must determine which blocks in the entire blockchain, starting at the genesis block, they are responsible for holding. Sharders begin storing new blocks immediately, and have some number of rounds to try to obtain older blocks for which they are responsible.

Clients who need to access an old block in the blockchain can request the block from one of the sharders responsible for holding it. A client might ask a sharder for a given block before that sharder has stored it (e.g., in case that sharder has only recently become active). In that case, the client can simply request the block from another sharder responsible for that block.

Blobbers. Blobbers provide storage for the 0Chain network. While they are primarily focused on facilitating decentralized applications (dApps), they are not limited to that role; if clients wish to use blobbers simply for distributed storage, they may do so.

Unlike miners and sharders, blobbers are not selected as part of the rotating active set. Instead, blobbers are assigned to handle a specific client's storage at the time that client first locks up tokens to acquire the right to storage resources. At that time, blobbers are selected by the mining network from the pool of available blobbers based on the needs of the 0Chain client. For instance, a 0Chain client might have business in London and San Jose, and hence blobbers located in those geographic regions would be preferable.

The 0Chain storage-protocol document [AMDB18] discusses blobbers in depth.

6 Advantages of 0Chain Consensus

Our protocol and its implementation incorporate several features that result in fast time to finality.

Faster network-traversal time. The time required for transaction propagation, block generation, verification, notarization, and finalization all scale linearly in BlockTime, which in turn is related to the network latency. Thus, decreasing the network latency directly impacts the time to finalization.

We keep the active set small, which lowers the latency required for messages to be propagated to all members of the active set. Note also that we do not require clients who are not currently part of the active set to receive messages sent as part of the consensus protocol during a given epoch; instead, clients will obtain a copy of the current head of the chain upon joining the active set.

Although we have a small active set, attacking the consensus protocol can be made to require an adversary to control *both* a majority of the active set *as well as* a majority of the stake that was staked by the members of the active set. By setting the mining reward sufficiently high, we can ensure that miners need to stake a large number of tokens in order to have a reasonable chance of controlling half the stake in the active set—something that would impose an unreasonably high cost on an attacker. Our squared staking approach also helps deter an attacker from being able to control more than half the stake in the active set.

We apply additional optimizations to reduce the network traffic, since this will also result in lower latency. For example, miners do not send full notarized blocks to the sharders; instead, they only send hashes of notarized blocks to the sharders, and the sharders then request any blocks they do not already have from a single miner.

Faster finalization. In executing the protocol we can dynamically adjust BlockTime based on the observed network latency in order to minimize the time to finalize blocks. In particular, we suggest to set BlockTime equal to the median (observed) network latency. In Appendix A we report on some experiments that illustrate the effect of setting BlockTime in this way.

Because we have a small active set, under honest operation the probability of having more than one finalized block in a given round is low.

References

[AMDB18] Thomas H. Austin, Paul Merrill, Siva Dirisala, and Saswata Basu. 0Chain storage protocol. Technical report, 2018.

[Aus18] Thomas H. Austin. 0Chain token reward protocol. Technical report, November 2018.

[ES18] Ittay Eyal and Emin Gün Sirer. Majority is not enough: bitcoin mining is vulnerable. *Communications of the ACM*, 61(7):95–102, 2018.

[HMW18] Timo Hanke, Mahnush Movahedi, and Dominic Williams. DFINITY technology overview series, consensus system. *CoRR*, abs/1805.04548, 2018.

[KLL$^+$97] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 654–663, New York, NY, USA, 1997. ACM.

[Nak09] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009.

[Poe15] Andrew Poelstra. On stake and consensus. https://download.wpsoftware.net/bitcoin/pos.pdf, 2015.

[Woo14] Gavin Wood. Ethereum: a secure decentralised generalised transaction ledger. https://gavwood.com/paper.pdf, 2014.

A Simulation Results

We simulated the 0Chain protocol with 100 miners, 4 generators, and 30 sharders (with each block being replicated 6 times). The miners are spread across 14 different data centers around the world (California, Canada, Frankfurt, Ireland, London, Mumbai, Ohio, Oregon, Paris, Seoul, Singapore, Sydney, Tokyo and Virginia). Although our simulations do not incorporate adversarial behavior, they do give a sense of how the protocol behaves under realistic network conditions. Our experiments provide empirical evidence regarding how finality time is related to the underlying parameters.

In our experiments, we found that the block-generation time and verification times are significant and comparable to message-delivery times; we also found that message-delivery times are noticeably different for short messages (e.g., VRF shares and verification tickets) and large messages (e.g., blocks), with the latter taking up to 3× longer to arrive than the former.

In our experiments, we vary the block-proposal wait time, wait mode (static vs. dynamic), and the finality wait time. The table below shows the time for block finality in steady-state, the start-to-finish time for block finality, and the percentage of rounds in which only one block was notarized. (In our experiments we never observed 4 blocks being notarized in a round, and the percentage of rounds in which 3 blocks were notarized never exceeded 0.03%.) In our experiments we set the finalization wait time to twice the network relay time, and set the finality lag to 3 blocks. Hence, the start-to-finish finality time is 3 times the steady-state block-finality time plus the finalization wait time.

Network relay time	BlockTime	Wait mode	Steady-state finality	Start-to-finish finality	% rounds, 1 notarization
200	200	static	1200 ± 140	4500 ± 300	95.1%
600	200	static	1300 ± 160	5300 ± 400	95.1%
400	400	static	1300 ± 140	5000 ± 400	95.3%
200	800	static	1400 ± 120	5000 ± 300	99.4%
200	200	dynamic	1300 ± 150	4600 ± 400	95.4%
400	400	dynamic	1300 ± 150	5000 ± 400	95.3%
200	800	dynamic	1300 ± 140	4600 ± 400	95.9%
200	1600	dynamic	1300 ± 200	4700 ± 400	97.0%

Table 1. Finality times as a function of the network and protocol parameters. All reported times are in milliseconds.