Proof-of-Approval: A Distributed Consensus Protocol for Blockchains

Shunsai Takahashi
shunsai.takahashi@gmail.com

February 13, 2018

Abstract

We present Proof-of-Approval protocol that, just like Nakamoto's famous blockchain protocol, enables achieving consensus in a so-called permissionless setting where anyone can join (or leave) the protocol execution. But unlike other protocols requiring consumption of physical resources, this protocol uses inherent randomness in network communication to arrive at a consensus. While all blockchains record transactions, this protocol additionally records approvals from network stakeholders. We show that recorded approvals make preferred fork selection less ambiguous, prevents long-range attacks and results in near instant finality.

This protocol allows anyone, including parties without stakes, to compete for the block creation process and win rewards. We show that this free-for-all approach results in high "liveness" for the blockchain. In addition, we show that the protocol provides design parameters to achieve the desired degree of "fairness."

We analyze the protocol in partially-synchronous "bounded-delay" model where the messages are guaranteed to arrive within the time bounds of a round. Our model also assumes the adversarial stake to be bounded below quorum and the total honest stake to be large enough to achieve quorum. Finally, we discuss some practical implications of this protocol.

1 Introduction

A distributed ledger is a database that lives on a network. It is shared, replicated and synchronized by its members without any trusted central authority. Just like a regular ledger, it records transactions such as exchange of assets. Transactions are never deleted from the ledger, only inserted. Members of the network, through consensus, decide what transactions are inserted in the ledger and what transactions are part of the ledger.

A blockchain is a distributed digital ledger that records transactions in a peer-to-peer network. All its records are held by its members and are stored on their own computer systems. Records of the blockchain are secured through

cryptographic hashes and then assembled into blocks which, in turn, are secured by cryptographic hashes. The cryptographic hashes of the blocks are also used to link future blocks sequentially to previous blocks, thus forming a chain like structure from the very beginning of that network. Since all transactions are securely linked from the beginning to the latest record, the ledger is fully auditable for each and every transaction.

Since there is no trusted central authority, creating consensus among a group of untrusted members becomes the central problem. Formally such a problem is called Byzantine Generals Problem [2]. Consensus ensures that all copies of the distributed ledger are exactly the same. It also lowers the risk of fraudulent transactions since tampering has to occur at all the copies at the same time. Cryptographic hashes ensure that even the minutest of the change in the content would result in a completely different hash flagging the contents as being potentially compromised.

A blockchain is considered only as secure and robust as its consensus protocol [3]. A blockchain can safeguard its key properties such as immutability and auditability only when its consensus protocol can uphold the state of the blockchain under physical failures as well as malicious intent. A poor choice of the consensus protocol will make a blockchain vulnerable to the attacks, compromising its data and rendering it useless. The next few sections discuss some of the most common consensus protocols.

1.1 Proof-of-Work (PoW)

Used by the two most valuable cryptocurrencies, it is one of the most popular algorithms. Any node can offer a candidate-block (and reap rewards) by solving a computationally difficult (by design) problem called mining. This consensus algorithm is susceptible to 2 types of attacks. First, "51% attack" where 51% of the mining capacity colludes to enter fraudulent transactions. Second, "selfish mining" [5] where normally honest nodes are incentivized to collude with the attacker.

In spite of these theoretical issues, this consensus algorithm has been used successfully by multiple blockchains for many years. It also scales well due to its simplicity of governance. Due to its consumption of resources, this algorithm can be costly for its members and may deter some from joining the network.

1.2 Proof-of-Stake (PoS)

Proof-of-Stake algorithm is designed to remove the resource consumption requirement from PoW algorithm by attributing mining power to the proportion of stake. Some PoS algorithms also take into account the age of the stake while others do not. Some PoS algorithms provide fresh coins for block creation [6] while others reward them only with transaction fees.

A naive PoS algorithm can suffer from several attacks including "stake grinding" and "nothing at stake" [7]. An implementation, PPCoin's PoS algorithm

also suffers from "rational forks," "bribe attacks" and "accumulated coin-age attack" [8]. At this time, a working PoS protocol is considered to be a non-trivial problem [11].

1.3 Hybrid Proof-of-Work/Proof-of-Stake

In light of issues raised by [11], Ethereum, a popular cryptocurrency, plans to continue using PoW for block creation but will add a PoS layer for block finalization [12].

1.4 Proof-of-Activity

A hybrid PoW/PoS approach [13] used by Decred. It consumes physical resources just like PoW and doesn't deter validators from double signing.

1.5 Proof-of-Burn (PoB)

The mining privileges are acquired by burning some coins (sending coins to an irretrievable address). Probability of mining a block is proportional to the amount of coins burned. It is used by slimcoin which is no longer active.

1.6 Proof-of-Elapsed-Time (PoET)

Intel proposed algorithm similar to PoW but without requiring large amounts of physical resources to be wasted. It requires trust in Intel which seems be contrary to the goals of public blockchains of not requiring trust in any third party. Such a system can be compromised by only a small fraction of participating nodes [14].

1.7 Ripple Protocol

Ripple uses collectively trusted subnetworks within the larger network to arrive at consensus [15]. A review of this protocol finds that the network is prone to disruption by as few as ten highly connected wallets [16].

1.8 Delegated-Proof-of-Stake (DPoS)

The DPoS algorithm elects a group of block-producers, who in turn, create blocks in their assigned timeslots. If a designated producer fails to produce blocks in its timeslots, it may lose its production privilege. Consensus is achieved by $\frac{2}{3} + 1$ majority of the producers [17]. Other protocols like Ouroboros [9] [10] and Tezos [22] also falls in the class of DPoS protocols.

Various versions of DPoS protocol are currently used by Steem, EOS, Cardano (Ouroboros) and Tezos.

1.9 Dfinity

Dfinity [24] registers all network nodes using an unspecified Sybil resistant method. During each timeslot, the network gives block creation ranks to all nodes in the network, the higher the rank the higher is the preference for the blocks they create. Other nodes, called notaries, sign the most preferred blocks and broadcast them to the network for faster finality and protection against long range revisions.

2 Proof-of-Approval

2.1 Summary

A network using this protocol would publish blocks periodically at a predefined interval. Each timeslot may create at most one block while some timeslots may not create any blocks. Each timeslot also specifies when certain events should occur within the slot.

This protocol is permissionless, i.e., parties (or nodes) are free to join and leave the network at will. Any node in the network, even those without any stake in it, are allowed to compete for block creation. The winning node is rewarded with fresh coins and transaction fees contained in the block.

Any node in the network can create a "candidate-block" and broadcast to the other nodes in the network. Each candidate-blocks is scored on how close to the target time it is received by a quorum of stakeholders. A candidate-block's score is a stake weighted function of its arrival time difference from the target time. Receiving nodes check for validity of the blocks and reject those containing anything invalid or leaving out valid transactions. Receiving nodes then rank valid candidate-blocks on how close to the target time each was received. Nodes also give the self-created candidate-blocks the top rank, providing stakeholders an inherent advantage in this competition. Receiving nodes then broadcast their approved list of candidate-blocks in descending order of preference to other nodes in the network.

A candidate-block's score is calculated from the ranks given by the receiving nodes. Each candidate-block's creator node is responsible for counting its candidate-block's score. If a candidate-block has a good score, its creator packages the received approvals in an "approval-block" and broadcasts it to the other nodes. Both the winning candidate-block and the approval-block are stored in the blockchain, and together, they are called a block.

In practice, the approval function will most likely be performed by the bigger stakeholders since their scores matter most. Although they are not paid anything for the approval, they are incentivized by their own increased chances of winning and their own interest in the smooth operation of the network.

If the network encounters communication difficulties, it will continue to function as normal as long as a quorum of the stakeholders can be reached. If not, the blockchain will freeze till the connectivity is restored.

2.2 Definitions and Components

2.2.1 Cryptographic Hash Function and Hash

A cryptographic hash function $h = H(m)$ has the following properties [19]

1. Pre-image resistance. Given hash value h, it is difficult to find m such that $h = H(m)$.
2. Second pre-image resistance. Given m_1, it is difficult to find m_2 such that $H(m_1) = H(m_2)$.
3. Collision resistance. It is difficult to find m_1 and m_2 such that $H(m_1) = H(m_2)$.

For the rest of this paper, we assume that the protocol uses a sufficiently secure cryptograhic hash function.

2.2.2 Asymmetrical or Public Key Cryptography

Matched pair of keys (K, K') for encode (E) and decode (D) functions can be found such that $D(E(m, K), K') = m$ [20]. Matched keys of the pair are called public and private keys respectively. They provide the following functionality

1. Neither K nor K' can be computed given the other.
2. Public key encryption. A message encrypted by public key K is readable only by the owner of K'.
3. Digital signature. A message encrypted by private key K' is readable by all and no one can impersonate the sender.

For the rest of this paper, we assume that the protocol uses sufficiently secure asymmetric key pairs for each party in the network.

2.2.3 Merkle Tree

A Merkle tree [21] is a data structure used to store transactions and approvals inside a block efficiently and securely.

2.2.4 Block, Content-Block and Approval-Block

Each block of a blockchain using this protocol consists of two parts, content-block and approval-block. Content-blocks store the transactions which is the main content of a blockchain. Approval-blocks store metadata for the blockchain that helps the network converge on the forks faster.

Nodes propose content-blocks that are called candidate-blocks till one among them is chosen to be placed in the blockchain.

2.2.5 Slot (Timeslot)

Time is divided into discrete units called slots. During each slot, at most one block (content-block and the corresponding approval-block) can be inserted into the chain. The following properties are assumed for the slots

1. Each party in the network has a roughly synchronized clock that indicates the current slot. Any discrepancies between the clocks are significantly smaller than the length of the time represented by a slot.

2. The length of the time represented by a slot is significantly larger than the time needed to transmit messages or blocks from one party to the other.

Each slot is represented by sl_r such that $r \in \mathbb{N}$. Each slot also defines

1. Target time slt_r to receive candidate-blocks. Candidate-blocks are scored on how close to the target time they are received by the receiving nodes with respect to their own clocks. A candidate-block received earlier than the target time is scored the same as if it were received the same amount of time after the target time.

2. Approval cutoff time sla_r is the deadline to receive approvals from the nodes on the network. Any approvals received beyond this time may not be used.

2.2.6 Parties (Nodes)

The network contains n parties (nodes) some of whom may be honest while others may be adversarial. Each party (node) owns a public and private key pair and is known to other parties on the network by its public key. The number of parties in the network may change at every timeslot.

A party is denoted as p_i where $i \in \{1, 2, .., n\}$ and its stake in the network as s_i.

2.2.7 Stake, Quorum and Maximum Exchange

Let T_r denote the total stake in the network at the beginning of slot sl_r. Decision making in the network requires a quorum stake denoted by $\rho \cdot T_r$. In each slot, the network allows only a fraction of the total stake to exchange hands. Let the maximum stake that can be transferred be denoted by $\delta \cdot T_r$. Each slot also allows creation of new stake which is denoted by $\nu \cdot T_r$. In practical implementations, minted stake fraction is significantly smaller than the maximum transfer fraction. Both of these are significantly smaller than the quorum fraction and the total stake.

$$\nu \ll \delta \ll \rho < 1 \tag{1}$$

2.2.8 Incentive System

We assume the following incentive system for the nodes in the network

1. Creators of blocks get freshly minted stake $\nu \cdot T_r$ that gradually decreases with increasing r.
2. Creators of blocks also collect all transaction fees of the transactions contained in the block.
3. Other participants, including approvers, do not receive any rewards.

2.2.9 Transactions

Transactions are the ledger entries that a distributed ledger like blockchain stores. A transaction is initiated by a node which then broadcasts it to other nodes. Each node maintains an updated list of transactions that are not yet stored in the blockchain. When a node creates a candidate-block, it uses this list to get transactions for it.

Each transaction is uniquely identified by its hash and can be validated by predefined rules. Invalid transactions are simply ignored and are never entered into the blockchain.

2.2.10 Content-Block

A block B_r holds the distributed ledger transactions recorded during the sl_r in its content-block. Figure 1 shows a typical content-block structure. Each

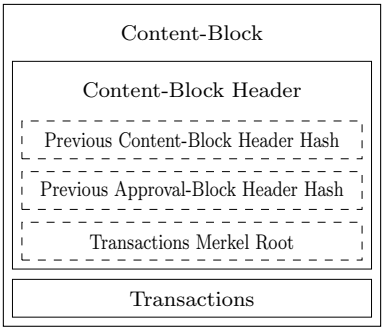

Figure 1: Content-Block Structure

content-block is uniquely identified by the hash of its header. Although not explicitly shown in the structure, each content-block header also includes timestamp from its creator.

2.2.11 Genesis Block

Genesis block is the very first block of a blockchain which is hardcoded in the software of the blockchain. It defines parties in the network at its onset and their corresponding stakes.

2.2.12 Content-Block Validation Predicate $V(\cdot)$

It returns 1 only when it is input with a content-block that is acceptable for the corresponding slot, 0 otherwise. This predicate is hardcoded to return 1 when input with the genesis block.

Content-block validation predicate checks for

1. Hash values of previous content-block and previous approval-blocks are correct and point to correct blocks.
2. Transactions are not empty if there are pending transactions in the network.
3. Each transaction included in the content-block is valid and not a duplicate of a past transaction.
4. The stake being transferred is less than $\delta \cdot T_r$.

2.2.13 Candidate-Block

A candidate-block c_{ri} is party p_i's content-block candidate for the slot sl_r. Each party receiving candidate-block c_{ri} must validate it through $V(\cdot)$ and reject the failing content-blocks. If a party p_i broadcasts more than one candidate-blocks for a specific slot, receiving nodes choose one of them as follows

1. Choose the candidate-block with the lowest (sender's) timestamp falling within the slot period.
2. If multiple candidate-blocks have the same timestamp, choose the block with the smallest header hash.

2.2.14 Approval-Message

During each slot, a node broadcasts at most one approval-message after receiving candidate-blocks from other nodes. Figure 2 shows a typical approval-message

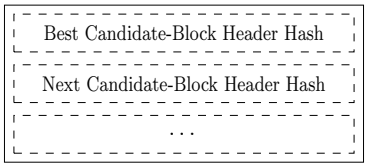

Figure 2: Approval-Message Structure

structure containing a sorted list of header hashes of approved candidate-blocks. The first hash corresponds to the most preferred candidate-block, followed by the hashes of other approved candidate-blocks in descending order of preference.

A node must validate received candidate-blocks through $V(\cdot)$ and reject the failing blocks. Each candidate-block is scored on how close to the target time slt_r it was received. Receiving a candidate-block before the target time generates the same score as if it were received the same amount of time after the target time slt_r. Each node then assembles approval-message and its list of preferred candidate-blocks in descending order of preference.

Each node puts its own candidate-block (if any) on top of its approved block list giving it an edge in the amount of its stake in the network. A node is also allowed to manipulate the order of the approved candidate-blocks but is unlikely to have an incentive for it. We discuss adversarial collusion below and show that combining stakes as one party would be preferable to colluding.

Each node then broadcasts its approval-message to other nodes. If a node p_i broadcasts more than one approval-messages for a specific slot, receiving nodes choose one of them as follows

1. Choose the approval-messages with the lowest timestamp falling within the slot period.

2. If multiple approval-messages have the same timestamp, choose the message with the smallest hash.

2.2.15 Approval Scoring Function $A(\cdot)$

Approval Scoring Function $A(\cdot)$ inputs the zero-indexed rank k ($k \in \mathbb{N}$, 0 being the best) of a candidate-block's header hash in the approval-message and outputs a score. It is a monotonic function such that

$$\forall \quad x, y \in \mathbb{N} \quad \text{and} \quad x \leq y, \quad A(x) \leq A(y) \tag{2}$$

Note that lower score is better.

2.2.16 Candidate-Block Scoring Function $G(\cdot)$

For each candidate-block, some nodes in the network send approval-messages, each with varying stake and approval rank. Score of a candidate-block is the smallest quorum stake weighted sum from the approval scores.

A node p_i may receive approvals from $\mathbb{J}$ nodes for its candidate-block c_{ri} such that $|\mathbb{J}| \leq n$. Each approving node $j \in \mathbb{J}$ has a stake s_j and approval score $a_j = A(rank_j)$. Let $\mathbb{A}$ denote the set of these approvals.

Function $G(\cdot)$ takes $\mathbb{A}$ and returns

1. If $\sum_{j=1}^{|\mathbb{J}|} s_j < \rho \cdot T_r$, return ∞.

2. Otherwise, sort $\mathbb{J}$ in ascending order of approval score and select first q nodes such that $\sum_{j=1}^{q-1} s_j < \rho \cdot T_r \leq \sum_{j=1}^{q} s_j$.

3. Return score $\sum_{j=1}^{q-1} a_j \cdot s_j + a_q \cdot \Big(\rho \cdot T_r - \sum_{j=1}^{q-1} s_j\Big)$.

Note that lower score is better.

2.2.17 Approval-Block

Each node collects approvals of its candidate-blocks and computes candidate-block's score relative to the other candidate-blocks. If the node's candidate-block's score is among the top, it broadcasts an approval-block to the network. Figure 3 shows a typical approval-block structure. Each approval-block

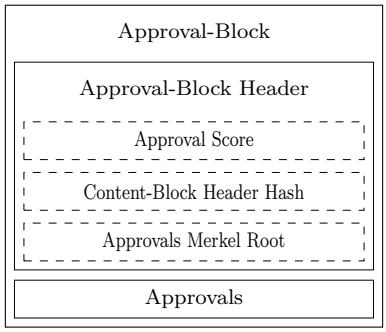

Figure 3: Approval-Block Structure

is uniquely identified by the hash of its header. Receiving nodes verify the approval-messages and approval score before accepting the approval-block.

If a node p_i broadcasts more than one approval-blocks, receiving nodes choose one of them as follows

1. Choose the approval-block with the lowest score received before the beginning of the next slot.

2. If multiple approval-blocks have the same score, choose the approval-block with the smallest hash.

If two approval-blocks have the same score, the approval-block with the smaller hash wins. After scoring approval-blocks, nodes add the winning approval-block and the corresponding content-block to their local chains. In subsequent timeslots, they will use hashes of these blocks in their candidate-blocks.

2.2.18 Block (Content-Block and Approval-Block) Scoring Function $S(\cdot)$

By definition, the genesis block has a score of 0. $S(\cdot)$ requires the stakes of the parties at the beginning of the slot. Stakes of the parties are calculated starting with the genesis block and walking up the blockchain assuming the fork under consideration is the consensus blockchain. This assumption is needed to break the implied recursion since the stakes depend upon the blockchain and the blockchain depends upon the stakes. The score is calculated as follows

1. If $S(\cdot)$ for any block between it and the genesis block returns ∞, return ∞.
2. If block's content-block fails $V(\cdot)$, return ∞.
3. Otherwise, return $G(\cdot)$ of its approval-block.

Note that lower score is preferred.

2.2.19 Blockchain and Fork

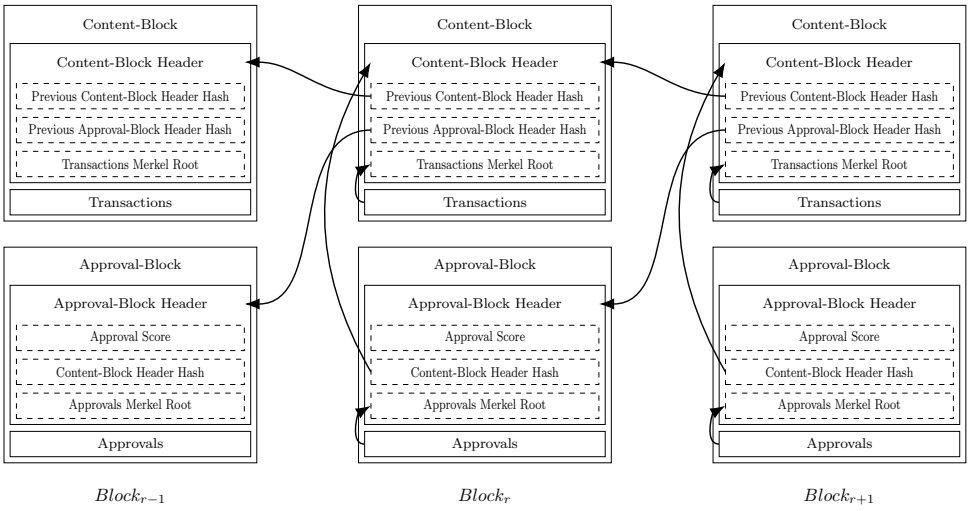

Figure 4: Proof-of-Approval Blockchain

For this protocol, a blockchain is a sequence of double-linked approval-blocks and content-blocks like Figure 4. For simplicity, we draw each block as a rectangle while fully understanding that the content-block and approval-block details are present underneath. At any instance a blockchain may look like Figure 5.

The rightmost (newest) block of a blockchain, is called "head" and is denoted by $head(\cdot)$. Figure 5 contains 2 blockchains, one whose head is block F while the other whose head is block E'. Note that both blockchains have the same

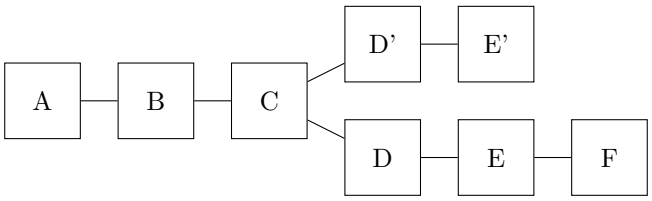

Figure 5: Blockchain and Forks

genesis block A as expected. They also share the blocks B and C while differing on blocks after C.

Length of a blockchain is its number of blocks. Blockchain whose head is F has length of 6.

2.2.20 Fork Comparison

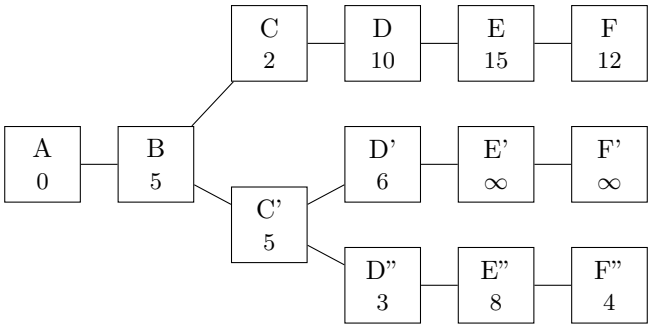

Figure 6: Fork Comparison

To determine the preferred blockchain among a set of forks, the following rules are applied starting from the genesis block.

1. For each slot, calculate the stakes at the beginning of the slot. Using $S(\cdot)$, score each block alternative for the slot. Any block scoring ∞ is invalid and is rejected along with all subsequent blocks in that fork.

2. If a slot contains multiple valid blocks, score subsequent blocks in each fork rejecting any block that score ∞ (and all subsequent blocks in that fork). In the end, the longest fork is the preferred fork.

3. If two or more forks have the same valid length, select the group of blocks of the junction closest to the genesis block. The fork with the block having the smallest score is the preferred fork.

Figure 6 shows forks along with the scores of their blocks. Block E' is invalid because its score is ∞. In addition, F' is invalid since it is located after

E'. Between forks (ending in) F' and F", fork (ending in) F" is preferred since it's longer than the other fork (whose last 2 blocks are invalid).

Between forks (ending in) F and F", fork F is preferred although both are the same length. It is because its block C scores better than C' where they fork. This example is contrived because once a quorum stake has accepted block D (as well as its parent C), there wouldn't be another quorum stake to accept block D" and give it a score lower than ∞. Note that blocks C or C' can change stake among the parties at most by $\delta \cdot T_r$ ensuring that no two forks have simultaneous yet distinct group of parties owning a quorum stake.

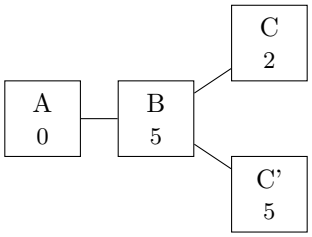

Figure 7: Temporary Fork

A fork in the very last block (e.g. Figure 7), is a temporary fork. Either of these forks may win after the subsequent blocks have been voted on. Once the network has decided to place blocks on C or C' and a majority agrees with one of them, that fork wins and the other disappears.

3 Analysis

3.1 Desiderata

In order to achieve a successful distributed ledger, a consensus protocol must provide the following properties [18][9].

3.1.1 Persistence, Weak Finality and Finality

Unlike centralized ledgers where a transaction is perpetual once recorded, a transaction recorded on a blockchain may not be as durable, at least not for some time immediately after recording. Durability of a transaction, called persistence, may be a probabilistic function of the number of blocks stored on the blockchain after the target transaction. Once a transaction goes more than k blocks "deep" into the blockchain of one honest party, then it will be included in every honest party's blockchain with an overwhelming probability.

Persistence can be restated as **Common Prefix** property. The blockchains maintained by the honest parties will possess a large common prefix.

Formally, for a parameter $k \in \mathbb{N}$, for any pair of honest parties p_1, p_2 adopting the chains C_1, C_2 at slots $r_1 \leq r_2$, it holds that $C_1^{\lceil k} \preceq C_2$ where $\preceq$ denotes prefix and $C_1^{\lceil k}$ denotes blockchain C_1 after removing rightmost k blocks.

Weak finality and finality are points on the persistence scale that may be useful for practical purposes. Weak finality can be considered a "good enough" durability of a transaction for e-commerce, e.g. shipping the goods after the payment has achieved weak finality. Finality, also called "strong consistency" [25], is durability comparable to the mainstream financial systems.

3.1.2 Fairness

Adversary should not be able to unfairly increase its chances of winning block creation.

It can be restated as **Chain Quality** property. The blockchain maintained by honest parties is guaranteed to have some blocks contributed by honest parties.

Formally, for parameters $\mu \in \mathbb{R}$ and $l \in \mathbb{N}$, for any honest party p with chain C, it holds that for any l consecutive blocks of C the ratio of adversarial blocks is at most μ.

We theorize that "ideal chain quality" for this protocol rewards blocks to parties approximately in proportion to their stakes in the network.

3.1.3 Liveness

Once an honestly generated transaction has been made available for a sufficient amount of time to the network nodes, it will become stable.

It can be restated as **Chain Growth** property. The blockchain maintained by honest parties grows at a minimum rate.

Formally, for parameters $\tau \in \mathbb{R}$ and $k \in \mathbb{N}$ for any honest party p with chain C, it holds that for any k rounds there are at least $\tau \cdot k$ blocks added to the chain of p.

3.2 Model

We define the following model for our analysis.

1. **Adversary**. The adversary controls some parties in the network, called corrupt parties, collectively represented by set $\mathbb{U}$. Let ϵ denote the corrupt fraction of the total stake such that

$$\epsilon \cdot T_r = \sum_{u=1}^{|\mathbb{U}|} s_u < (\rho - \delta - \nu) \cdot T_r \tag{3}$$

$$\rho > \epsilon + \delta + \nu \tag{4}$$

   Parties $\mathbb{U}$ can perfectly synchronize their behavior to achieve the maximum adversarial impact.

2. **Network Communication**. The network is synchronous in the sense that an upper bound can be determined during which any honest party

is able to communicate with any other party. Also, this upper bound is much smaller than the slot time period.

We also assume that parties are connected through equally performant network connections and the adversary is not able interfere with communications originating from the honest parties.

3. **Messages**. The adversary may be able to spoof the originating node of a message, but is not able to spoof the digital signature of the originating party. Therefore, a digitally signed message is guaranteed to originate from the signing party.

   The adversary also does not posses capability to find different contents producing the same cryptographic hash. Therefore, a cryptographic hash of a message is guaranteed to represent the original unaltered message.

4. **Approval Scoring Function**. Our model assumes approval scoring function to be
$$x \in \mathbb{N}, \quad A(x) = x \tag{5}$$

5. **Maximum Single Party Stake**. No single party or colluding parties hold a quorum stake in the network.

6. **Quorum Stake Fraction**. We choose ρ near but greater than 0.5 such that honest stake $1 - \epsilon > \rho$.

3.3 Evaluation

3.3.1 Common Prefix

Theorem 3.1 (Common Prefix)**.** *In a typical execution, the common-prefix property holds with parameter $k \geq 1$.*

Proof. Figure 8 shows a blockchain having a single fork up till slot sl_{r-1} having a block B_{r-1} at its head. At slot sl_r, many blocks are proposed denoted by blocks B_r, $\dot{B}_r$, $\ddot{B}_r$ and others. Each of the proposed block will include a newly minted stake of $\nu \cdot T_r$ and a transaction amounts up to a maximum of $\delta \cdot T_r$. Let us call a candidate-block and its subsequent blocks, which may or may not persist, a candidate-fork.

Let us assume the worst case scenario where each of the candidate-blocks for sl_r include transactions of the maximum amount $\delta \cdot T_r$. Further assume that such transactions in the different candidate-blocks go to different parties altering the stakes among the candidate-forks. In other words, each candidate-fork will have a slightly different stake ownership in the network. Newly created stake $\nu \cdot T_r$ will also modify stakes in the candidate-forks. For the very worst case scenario, let us assume that the newly created stake as well as the transferred stake goes to parties that had no prior stake in the network and this stake is transferred only from the honest parties (not from adversary).

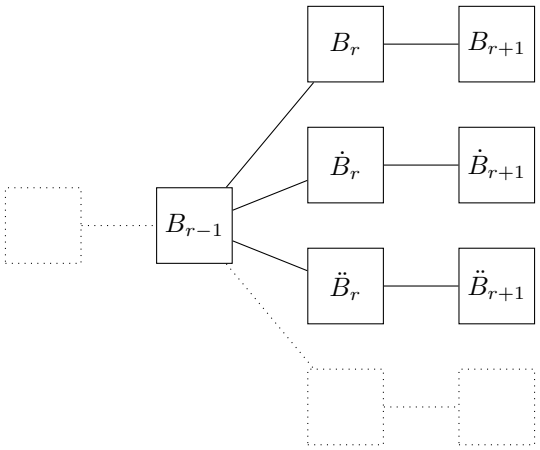

Figure 8: Common Prefix

For the slot sl_r, honest parties as well as the corrupt parties will approve and prioritize candidate-blocks. Since adversary controls less than a quorum stake, at least some honest stake is required for a candidate-block to have a score below ∞. Therefore, each candidate-block in candidate-fork must pass validation of at least one honest node. In addition, the honest parties will make their preference among these blocks based upon their scores.

For the slot sl_{r+1}, stakes are slightly different for each candidate-fork. Blocks B_{r+1}, $\dot{B}_{r+1}$, $\ddot{B}_{r+1}$, must receive approval from a quorum stake in their respective candidate-fork in order for their candidate-fork to persist. In each candidate-fork at slot sl_{r+1}, the total stake is

$$T_{r+1} = (1 + \nu) \cdot T_r \tag{6}$$

Honest parties' stake remaining in any candidate-fork, denoted by H_{r+1} (after subtracting δ transferred stake), is

$$H_{r+1} = (1 - \epsilon - \delta) \cdot T_r \tag{7}$$

The stake that would vote unconditionally for a candidate-fork, denoted by A_{r+1}, is

$$A_{r+1} = T_{r+1} - H_{r+1} = (\epsilon + \delta + \nu) \cdot T_r \tag{8}$$

The quorum stake in any candidate-fork is

$$\rho \cdot T_{r+1} = \rho \cdot (1 + \nu) \cdot T_r \tag{9}$$

From Equation 4 and noting that ν is positive, quorum stake at any candidate-fork

$$\rho \cdot T_{r+1} > \rho \cdot T_r > (\epsilon + \delta + \nu) \cdot T_r = A_{r+1} \tag{10}$$

Even if all corrupt parties approve multiple candidate-blocks, and newly acquired stakeholders vote for their candidate-fork, at least one honest party is

still required to achieve quorum. Since the honest parties would have chosen a candidate-fork based on the scores of blocks in sl_r, they will approve only one candidate-fork. Therefore, only one of the B_{r+1}, $\dot{B}_{r+1}$, $\ddot{B}_{r+1}$ etc. will receive a score below ∞ and, therefore, there will be only one persistent fork. This proves that the common prefix property holds for $k \geq 1$. □

Theorem 3.2 (Weak Finality and Finality)**.** *In a typical execution, weak finality and finality are achieved with parameter $k \geq 1$.*

Proof. Theorem 3.1 shows that all honest parties have the common chain prefix for $k \geq 1$. Therefore, any transaction in a block buried by one or more blocks is held by all chains of all honest parties. Therefore, any honest party will report that transaction after one or more blocks have been deposited on top of the block containing the target transaction. □

3.3.2 Chain Quality

Theorem 3.3 (Zero Stakes)**.** *Parties with zero stakes in the network have non-zero chances of creating a winning block.*

Proof. Each party is allowed to rank their own block on top which carries weight of their own network stake. If a party p_i has s_i stake in the network at slot sl_r, then it needs to acquire approvals from holders of the remaining $\rho \cdot T_r - s_i$ stake. Also note that the combined stake of the honest parties in the network is larger than the quorum stake. It is easy to see that while own stake helps to promote own candidate-block, nodes with zero stake still have sufficient honest parties that can make their candidate-blocks win. □

Theorem 3.4 (Approval Scoring Function)**.** *If there are two corrupt parties on the network with stakes s_i and s_j, their preference on collusion (operating as 2 parties) vs. combination (pooling stakes into a single party) depends on the approval scoring function $A(\cdot)$.*

Proof. Consider the following approval scoring function

$$A(x) = \begin{cases} 0 & \text{if } x \in \{0, 1\} \\ x & \text{if } x \in \{2, 3, \ldots\} \end{cases} \tag{11}$$

In this case, the colluding parties (that give each other the second best preference) will have statistically as good a score as if they were combined. But in collusion case, they have 2 candidate-blocks while in combined case, they have only one block with only half as much chances of winning. This scoring function would make the corrupt parties want to collude.

On the other hand, consider the following approval scoring function

$$A(x) = \begin{cases} 0 & \text{if } x = 0 \\ 1000 + 0.1x & \text{if } x \in \{1, 2, \ldots\} \end{cases} \tag{12}$$

This scoring function puts severe penalty for the second spot. If the parties were to combine, their combined block is likely to get a lot better score and much better chances of winning than their two separate blocks. This scoring function would make corrupt parties want to combine instead of collude. □

Theorem 3.5 (Reducing Winning Chances of Adversary)**.** *A properly chosen approval scoring function can reduce chances of the adversary winning unfairly.*

Proof. As shown by the previous theorem, approval scoring function can alter the reward ratio for combining vs. colluding. We theorize that it is more difficult for the corrupt parties to combine than to collude. Choosing an approval scoring function that favors combining, will reduce winning chances of colluding adversaries. □

3.3.3 Chain Growth

Theorem 3.6 (Chain Growth)**.** *In a typical execution the chain-growth property holds with parameters* $\tau = 1$.

Proof. At every round, multiple parties create candidate-blocks and broadcast them to other parties. Since honest majority consists of multiple parties (no single party holds a quorum stake), they will broadcast valid blocks to the network. Even if corrupt parties do not approve any of these candidate-blocks, the honest stake has quorum and continues to approve candidate-blocks. □

4 Practical Implications

4.1 Network Connectivity Centric

The protocol we present is clearly network connectivity centric. Nodes with better network connections are likely to reach other parties before those with poorer network connections, thereby increasing their chances of winning. A typical mobile network or even a home or office network is far too slow compared to a cloud network. Since the protocol doesn't require large computations, just fast connectivity, a small cloud instance with good connectivity is likely to suffice. Such small instances can be acquired for approximately $5/month.

4.2 Active and Passive Parties

For many of the network participants, even the amount of $5/month may be too much to spend. Those parties will likely not participate in the block creation process. Therefore, participants will self select into two groups, "active" parties that maintain a good network connectivity and try winning block creation, and "passive" parties that just use the network for transactions.

4.3 Additional Storage Requirements

The protocol presented here uses more storage than many other protocols because it stores metadata (approval-messages) in addition to the transactions. This will increase the storage size of the blockchain compared to other protocols. Since the cost of the storage for a few more GB is fairly small, we believe this does not present a prohibitive barrier.

4.4 Scalability

Proof-of-Approval scales well for the following reasons:

4.4.1 Pareto Stake Distribution

Stake distribution among parties is likely to be Pareto distribution where most of the network stake is concentrated among a few hundred nodes. It is these high stake holders that would be doing most of the validation and ranking of the candidate-blocks.

Candidate-block creators, even when they are running on a cloud instance with excellent connectivity, still have to prioritize the target nodes for their candidate-blocks limiting the total messages sent during each timeslot.

Majority of the nodes in the network will self select themselves into passive mode to save on cloud server costs. These nodes will neither create candidate-blocks nor rank them.

4.4.2 Timeslot Period

Timeslot period should be kept small enough to let nodes having better network connections stand out, which in turn would result in a higher transaction throughput.

Keeping timeslot smaller provides an incentive for nodes to strive for faster and more reliable network connections. As a result, a larger fraction of network stake would be available online making the blockchain operation smoother.

References

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

[2] L. Lamport, R. Shostak, M. Pease, *The Byzantine Generals Problem*, ACM Transactions on Programming Languages and Systems, 4(3), 1982

[3] M. Conti, C. Lal, S. Ruj et al., *A Survey on Security and Privacy Issues of Bitcoin*, eprint arXiv:1706.00916, 2017

[4] J. Douceur, *The Sybil Attack*, International Workshop on Peer-to-Peer Systems, March 2002

[5] I. Eyal, E. G. Sirer, *Majority Is Not Enough: Bitcoin Mining Is Vulnerable*, Financial Cryptography, 2014

[6] S. King, S. Nadal, *PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake*, 2012

[7] V. Buterin et al., *Proof of Stake FAQ*, Ethereum Foundation, `https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ`, 2018

[8] I. Bentov, A. Gabizon, A. Mizrahi, *Cryptocurrencies without Proof of Work*, Proceedings of the 2016 Financial Cryptography and Data Security Conference, 2016

[9] A. Kiayias, A. Russell, B. David et al., *Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol*, Cryptology ePrint Archive, Report 2016/889, 2016

[10] D. Larimer, *Peer Review of Cardano's Ouroboros*, `https://steemit.com/cardamon/dan/peer-review-of-cardano-s-ouroboros`, 2017

[11] V. Buterin, *Slasher Ghost, and Other Developments in Proof of Stake*, `https://blog.ethereum.org/2014/10/03/slasher-ghost-developments-proof-stake/`, 2014

[12] V. Buterin, V. Griffith, *Casper the Friendly Finality Gadget*, eprint arXiv:1710.09437, 2017

[13] I. Bentov,C. Lee, A. Mizrahi et al., *Proof of Activity: Extending Bitcoin's Proof of Work via Proof of Stake*, Cryptology ePrint Archive, Report 2014/452, 2014

[14] L. Chen, L. Xu, N. Shah et al., *On Security Analysis of Proof-of-Elapsed-Time*, 19th International Symposium on Stabilization, Safety, and Security of Distributed Systems, 2017

[15] D. Schwartz, N. Youngs, A. Britto, *The Ripple Protocol Consensus Algorithm*, `https://ripple.com/files/ripple_consensus_whitepaper.pdf`, 2014

[16] P. Moreno-Sanchez, N. Modi, R. Songhela et al., *Mind Your Credit: Assessing the Health of the Ripple Credit Network*, eprint arXiv:1706.02358, 2017

[17] D. Larimer, *DPOS Consensus Algorithm - The Missing White Paper*, `https://steemit.com/dpos/@dantheman/dpos-consensus-algorithm-this-missing-white-paper`, 2017

[18] J. Garay, A. Kiayias, N. Leonardos, *The Bitcoin Backbone Protocol: Analysis and Applications*, `http://eprint.iacr.org/2014/765`, 2014

[19] E. Andreeva B. Preneel, *A Three-Property-Secure Hash Function*, Selected Areas in Cryptography, 2008

[20] G. Simmons , *Symmetric and Asymmetric Encryption*, ACM Computing Surveys, 11(4), 1979

[21] R. Merkle, *A Digital Signature Based on a Conventional Encryption Function*, Advances in Cryptology, 1987

[22] *Tezos: The self-amending cryptographic ledger*, `https://www.tezos.com/static/papers/Tezos_Overview.pdf`, 2017

[23] *Proof-of-stake in Tezos*, `http://doc.tzalpha.net/whitedoc/proof_of_stake.html`, 2018

[24] T. Hanke, M. Movahedi, D. Williams, *DFINITY Technology Overview Series: Consensus System*, DFINITY Technology Overviews, 2018

[25] S. Bano, A. Sonnino, M. Al-Bassam et al., *Consensus in the Age of Blockchains*, eprint arXiv:1711.03936, 2017