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

Shunsai Takahashi
shunsai.takahashi@gmail.com

April 23, 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 Nakamoto's protocol and others, this protocol does not require consumption of physical resources. Instead, this protocol uses explicit approvals from stakeholders and network randomness to arrive at a consensus. A blockchain using this protocol records these explicit approval messages in addition to recording the transactions. We show that this protocol achieves near instant finality and deters attacks that plague other stake based protocols.

This protocol allows anyone with a minimum stake to compete for the block creation and win rewards and associated transaction fees. At any slot, multiple block creators compete to create the latest block resulting in high "liveness" for the blockchain. Each block must be approved by a quorum stake for it to be valid and placed in the chain. In addition to block approvals, this protocol incorporates epoch approvals to deter Costless Simulation attacks. Block approvals as well as epoch approvals are non-competitive and all approvers are paid in proportion to their stakes in the network.

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

Revision	Date	Author(s)	Description
1.0	February 13, 2018	ST	Initial version
2.0	April 23, 2018	ST	Protocol and reward system updates

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 the 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 most of 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 attacks, compromising its data and rendering it useless.

1.1 Extrinsic vs. Intrinsic Opportunity Cost

Bitcoin [1] uses an extrinsic opportunity cost known as Proof-of-Work [4]. Any extrinsic opportunity cost function necessarily consumes physical resources. In order to remove the physical resource consumption requirement from the consensus, intrinsic opportunity cost functions, using the blockchain itself (typically the stakes of the parties), have been proposed. Various Proof-of-Stake protocols and their derivatives are examples of consensus using such intrinsic opportunity cost functions.

It has been argued that consensus using an intrinsic opportunity cost function

is inherently flawed and cannot be made to work [5] [6]. Furthermore, such consensus protocols suffer from the following attacks [7] [8].

1.1.1 History or Costless Simulation Attack

In this attack [5], also called Long Range attack [8], an attacker acquires account information (private keys) of one ore more parties that held a large amount of stake in the past (but may not hold any at present). Using these accounts, the attacker creates blocks from some time in the past to rewrite the blockchain history in his favor.

Depending upon the consensus protocol, such an attack typically requires accounts representing a majority stake in the network at some point in time. If the accounts presently hold none or little stake in the network, their owners may have little incentive to keep those keys private and an attacker may be able to acquire them relatively inexpensively.

1.1.2 Nothing at Stake Attack

In this attack [8] [10], if there is a fork of the blockchain, accidental or deliberate, the rational behavior for all parties in the network is to build blocks on all branches. This makes it easy to perform double-spending or other sorts of attacks relying on forking the blockchain. As long as users of the system believe the attack may succeed, they will support it by building on top of the attacker's blockchain. While forking attacks will be opposed by users with a large amount of stake who will fear to lose their money, if the stake is evenly distributed among many users, the attack is more likely to succeed.

1.1.3 Bribe Attack

In this attack [7], the attacker is able to double spend by bribing other parties in the network. After the merchant waits for e.g. 6 block confirmations and sends the goods, the attacker can publicly announce their intent to create a fork that reverses the last 6 blocks, and offer bribes to stakeholders who would sign blocks of their competing branch that starts 6 blocks earlier. If the attacker offers total bribe smaller than the double spend amount, neither the attacker nor the colluders have any risk. They all win at the merchant's expense.

1.1.4 Stake Grinding Attack

This attack [9], also known as Precomputing attack [10], applies to networks where the right to create blocks is limited by some random functionality. In

this attack, the attacker performs some computation or takes some other steps to try to bias the randomness in their own favor. As a result, the attacker gains unfair advantage and rewards.

1.1.5 Stake Bleeding Attack

This attack [11] leverages transaction fees, the ability to treat transactions "out of context" and the standard longest chain rule to completely dominate a blockchain. The attack grows in power with the number of honest transactions and the stake held by the adversary, and can be launched by an adversary controlling any constant fraction of the stake.

2 Proof-of-Approval

2.1 Summary

A network using this protocol publishes blocks periodically at a predefined interval called slot. Each slot may create at most one block while some slots may not create any blocks. The protocol defines another larger interval containing a predefined number of slots, called epoch.

This protocol is permissionless, i.e., parties (nodes) are free to join and leave the network at will. For every slot, several nodes in the network having a minimum stake, are allowed to compete for block creation. Creator of the winning block is rewarded with fresh coins and transaction fees contained in the block.

A block must be approved by stakeholders holding a quorum of stake for it to be valid and placed in the blockchain. Block creators broadcast their blocks on the network and the receiving nodes validate and optionally approve the blocks. Nodes indicate their approval through an explicit message to the block creator. Approvers are allowed to approve as many blocks as they choose as long as the approved blocks share the same parent. If not, the approvals are considered conflicting and cannot be used. Approvers (with valid approvals) are rewarded in proportions to their stake in the network.

When the approvals exceed the required quorum stake, the block creators broadcast the collected approvals to the network. Creators of the next block would place these approvals inside the blocks they create. The approval stake stored inside a block also determines the owners of which coin rolls are allowed to create the next block. For every slot, the block creators build on the block containing the highest stored approval stake.

The last block of an epoch is special in the sense that it contains epoch approvals. Epoch approvals are similar to block approvals but are designed to receive approvals from nodes on slow or intermittent connections. Epoch ap-

provers are awarded about the same as the block approvers but, unlike block approvals, requires little computation or network connectivity. Epoch approvals encourage all nodes, even those with limited computational or network capacity, to sign and approve the chain. Epoch approvals deter History attacks.

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, many slots will not create any blocks until the connectivity improves.

2.2 Definitions and Components

2.2.1 Glossary

1. $A(\cdot)$. Approval tie-breaking procedure.
2. B_r. A block at slot number $r \in \mathbb{N}$.
3. C. A chain or fork.
4. k. Number of blocks needed on top of a block for stability (for Common Prefix).
5. m. Number of slots in an epoch.
6. $\mathbb{N}$. Set of natural numbers, 0, 1, 2, ...
7. n. Number of parties in the network.
8. p_i. A party where $i \in \{1, 2, .., n\}$.
9. $R(\cdot)$. Coin roll selection predicate.
10. $\mathbb{R}$. Set of real numbers.
11. s_i. Party p_i's network stake where $i \in \{1, 2, .., n\}$.
12. sl_r. A slot where $r \in \mathbb{N}$.
13. T_r. Total stake (not fraction) at the beginning of slot number $r \in \mathbb{N}$.
14. $\mathbb{U}$. Set of corrupt (adversarial) parties.
15. $V(\cdot)$. Block validation predicate.
16. δ. Maximum stake fraction that can be transferred in one block, includes transaction fees.
17. ϵ. Adversarial stake fraction.
18. μ. Fraction of blocks created by adversarial parties (for Chain Quality).
19. ν_a. New stake fraction allocated for block approvers.
20. ν_c. New stake fraction given to the winning block creator.
21. ν_e. New stake fraction allocated for epoch approvers.
22. ρ. Quorum stake fraction used for decision making in the network.
23. τ. Fraction of blocks created per slot (for Chain Growth).

2.2.2 Cryptographic Hash Function and Hash

A cryptographic hash function $h = H(x)$ has the following properties [15]:

1. Pre-image resistance. Given hash value h, it is difficult to find x such that $h = H(x)$.
2. Second pre-image resistance. Given x_1, it is difficult to find x_2 such that $H(x_1) = H(x_2)$.
3. Collision resistance. It is difficult to find x_1 and x_2 such that $H(x_1) = H(x_2)$.

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

2.2.3 Asymmetrical or Public Key Cryptography

Matched pair of keys (K, K') for encode (E) and decode (D) functions can be found such that $D(E(m, K), K') = m$ [16]. 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.

2.2.4 Merkle Tree

A Merkle tree [17] is a data structure used to store transactions and approvals inside a block efficiently and securely.

2.2.5 Slot

Time is divided into discrete units called slots. During each slot, at most one 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 $r \in \mathbb{N}$.

2.2.6 Epoch

The protocol defines a larger interval, epoch, that contains m consecutive slots. Note that epochs may have fewer blocks than slots due to one or more slots not inserting a block in the chain.

Each epoch stores, in one or more of its blocks, one or more pieces of slowly changing information. The purpose of the epochs is purely to improve the storage efficiency of the blockchain.

2.2.7 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 slot.

A party is denoted as p_i where $i \in \{1, 2, .., n\}$ and its stake in the network as s_i. We expect stake distribution to follow Pareto distribution where a large portion of the stake is held by a small portion of the nodes.

We expect parties to have varying network connection speeds, from high-speed (>1Gbps) cloud connectivity to low-speed (<1Mbps download/100Kbps upload) connections.

2.2.8 Coins and Rolls

The nominal unit of the network stake is a coin and the smallest transaction amount is a very small fraction of a coin. A roll contains a predefined number of coins which typically is set to many thousands.

Each roll is assigned an id which may be similar to the public keys of the nodes. The purpose of assigning ids to rolls, and not to coins, is purely to improve the storage efficiency of the blockchain. When a roll is broken up for a transaction, its id is destroyed. Alternatively, coins that are not yet part of a roll, can be assembled into a roll which is assigned a new id upon creation.

The total network stake is thousands of times larger than the stake represented by a roll and does not have to be its multiple.

2.2.9 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. This exchange fraction include transaction fees for all transactions contained in a block. Let this be denoted by $\delta \cdot T_r$.

In practical implementations, maximum transfer fraction is significantly smaller than the quorum fraction and the total stake.

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

2.2.10 Incentive System

Creators of blocks are awarded freshly minted stake $\nu_c \cdot T_r$. Creator of a block also collects all transaction fees for the transactions contained in the block.

Approvers of blocks are distributed a maximum of $\nu_a \cdot T_r$ stake such that each approver receives $\nu_a \cdot T_r \cdot s_i$ stake. Note that the creator of the block also gets approval award.

Approvers of epochs are distributed a maximum of $\nu_e \cdot T_r$ stake such that each approver receives $\nu_e \cdot T_r \cdot s_i$ stake per epoch. Epoch approval awards are designed for maximum stakeholder participation providing significant award for very little effort. Even the stakeholders on the slowest connection would easily be able to win epoch approval awards.

While the block creator award along with transaction fees is competitive, the block approval and epoch approval awards are non-competitive. All award fractions, ν_c, ν_a, ν_e, gradually decreases with increasing r.

In practical implementations, block creator award fraction ν_c is significantly smaller than block approval award fraction ν_a which approximately equals epoch approval award fraction divided by the number of slots in an epoch.

$$\nu_c \ll \nu_a \approx \nu_e/m \ll \delta \tag{2}$$

2.2.11 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 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.

Transactions include hash of a recent block and therefore are "context sensitive" [12]. This prevents transactions from being used in an attack fork.

2.2.12 Block

A blockchain stores its information in blocks. Figure 1 shows a typical block structure for a chain using this protocol. Each block is uniquely identified by the hash of its header. Although not explicitly shown in the structure, each block header also includes the slot number it is targeted for.

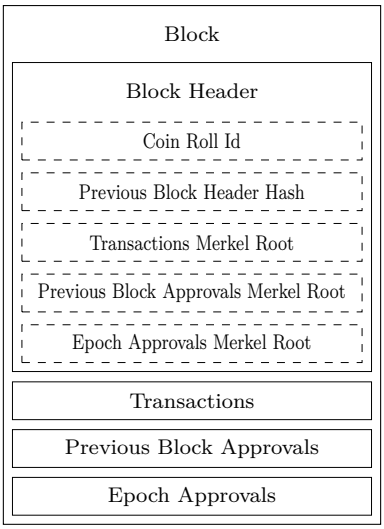

Figure 1: Block Structure

A block B_r holds the distributed ledger transactions recorded during the sl_r as well as approvals for the B_{r-1}. The last block in an epoch also holds epoch approvals and is called epoch block.

2.2.13 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, their corresponding stakes and the ids of the coin rolls.

The genesis block needs no approvals, therefore, the first block after the genesis block does not include approvals.

2.2.14 Block Creator Selection and Candidate Blocks

In each slot, multiple (approximately 10-50) stakeholders are allowed to create candidate blocks. Each party must run the coin roll selection predicate $R(\cdot)$, with ids of coin rolls it owns, to determine if it is allowed to create a block in that slot.

Having more than one candidate block per slot results in high liveness for the chain. Also, limiting the number of the candidate blocks prevents the communications from being bogged down due to traffic.

2.2.15 Coin Roll Selection Predicate $R(\cdot)$

It takes the following inputs and returns 1 if the roll owner can create block, 0 otherwise:

1. Total approval stake (rolls, coins and fractional coins) stored in the block that the new block is pointing to. The algorithm uses the coins and the fractional coins of the approval stake since they are more random than the total stake.
2. The current slot id. It ensures that even if the same group of parties approves multiple blocks in a row, a different set of block creators are chosen for each slot.
3. Total number of coin rolls.
4. Number of parties n.
5. Coin roll id.

2.2.16 Block Approval Message

Nodes, after receiving candidate blocks, validate them through the block validation predicate $V(\cdot)$ and optionally send their approvals, in form of an approval message, directly to the creator of the candidate block. Figure 2 shows a typical approval message structure containing signed header hash of the approved candidate blocks.

Nodes are allowed to approve as many candidate blocks as they choose as long as they share the same parent block. In Figure 3, a node can choose to approve either D by itself or D' and D" for the slot. But if it were to approve D as well as D', since those don't share the same parent, those approvals would be called "conflicted" and are rejected during validation.

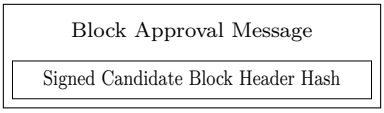

Figure 2: Block Approval Message Structure

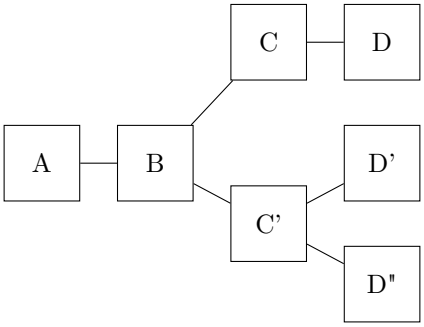

Figure 3: Acceptable and Conflicted Block Approvals

2.2.17 Block Approval Block

Each node collects approvals of its candidate block and computes its approval stake. Any approval with conflict or zero stake is ignored. If the node's candidate block's approval stake exceeds quorum, it broadcasts an approval block to the network.

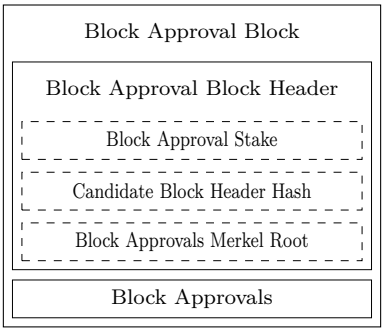

Figure 4: Block Approval Block Structure

Figure 4 shows a typical approval block structure. Receiving nodes collect all approval blocks and validate each and check if any approvals are in conflict. A node can broadcast an improved version of its approval block when it receives additional approvals or when it detects approval conflict. Receiving nodes use the approval block with the most stake to create their candidate block for the

next slot.

Since the approval stake determines the next block creator, a block creator may be inclined to keep some approvals out so that the approval stake allows them to create the next block (in addition to the current block). That strategy does not work since approvers are allowed to approve multiple blocks (from the same parent) and another block may have gotten equal or higher approval stake.

2.2.18 Epoch Approvals

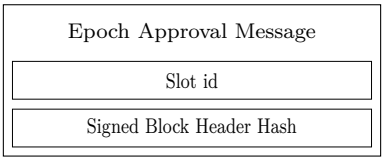

Figure 5: Epoch Approval Message Structure

An epoch approval message is a signed header hash of any block in the epoch, typically the very first block. Each node broadcasts this message to other nodes in the network. Each node also maintains an updated list of received valid epoch approvals during the course of an epoch. When a node creates a candidate block for the last block of an epoch, it includes epoch approvals from this list in that block.

Parties are allowed to approve only a single block for epoch approvals. If a party sends epoch approvals for more than one blocks for an epoch, the block belonging to the higher slot is selected. If a party approves multiple blocks of the same slot, all such approvals are considered invalid. Epoch approvals with zero stakes are ignored.

2.2.19 Block Validation Predicate $V(\cdot)$

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

Block validation predicate checks for:

1. The coin roll id is valid for creating block given the previous block.
2. The block creator owns the coin roll with the specified id.
3. Hash value of the previous block points to the correct block.
4. Previous block approvals and the Merkel root are valid.

5. No approval is in conflict.
6. Total approver stake is at least a quorum stake.
7. If the block is the last block of an epoch, it contains epoch approvals representing at least a quorum stake.
8. If the block is the last block of an epoch, no epoch approvals are invalid.
9. Transactions are not empty if there are pending transactions in the network.
10. Each transaction included in the block is valid and not a duplicate of a past transaction.
11. The stake being transferred is less than $\delta \cdot T_r$.

2.2.20 Choosing Blocks for Approval

Nodes want to maximize their block approval award. Since nodes can approve multiple candidate blocks having the same parent, the block choice boils down to choosing a parent block that has the most chances of winning. Typically this would be the block containing the most approval stake. Once the parent is chosen, the nodes would simply approve all valid children candidate blocks.

2.2.21 Choosing a Block to Extend

Again, nodes want to maximize their block creation award. Since a block with maximum stored approvals is preferred by approvers, they would choose a block that has received the maximum approval stake. Nodes may want to create a new block if the another candidate containing higher approval stake is received.

2.2.22 Approval Tie-Breaking Procedure $A(\cdot)$

It is possible to have multiple blocks with exactly the same approval stake, even down to the smallest fractional coins. In that case, the complete block selection procedure becomes:

1. Block with the higher approval stake, even to the smallest fractional coins, wins.
2. If the approval stakes match exactly, then the block with fewer approving parties wins. This discourages parties to split their stakes in multiple accounts.
3. If approval stakes match and the number of parties match, the block containing the smallest approval signature wins. While the approval signatures are not manipulatable, a party may benefit by splitting its stake

into multiple accounts. But the previous step would discourage such an action.

2.2.23 Blockchain and Fork

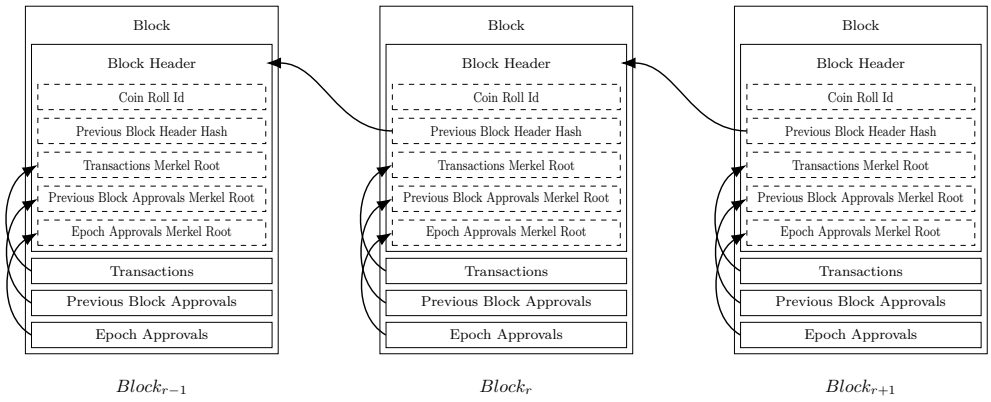

Figure 6: Proof-of-Approval Blockchain

For this protocol, a blockchain may look like Figure 6. For simplicity, we draw each block as a rectangle while fully understanding that the other details are present underneath. At any instance a blockchain may look like Figure 7.

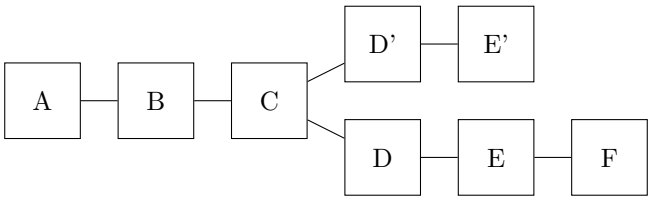

Figure 7: Blockchain and Forks

The rightmost (newest) block of a blockchain, is called "head" and is denoted by $head(\cdot)$. Figure 7 contains 2 chains (or forks), one whose head is block F while the other whose head is block E'. Note that both chains have the same genesis block A as expected. They also share the blocks B and C while differing on blocks after C.

2.2.24 Chain Length

The chain length specifies the slots spanned by the chain, not the number of blocks contained in it. For example, in Figure 8, chain having A_6 has length 6 while chain having B_5 has length 5.

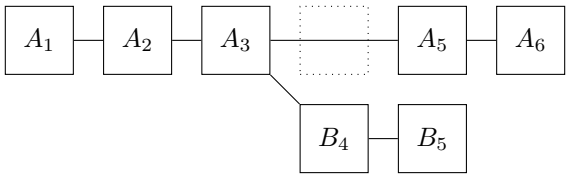

Figure 8: Chain Length

This chain length method is designed for the real life chains that are likely to miss blocks due to parties not being online or poor network connectivity. Attack chains, on the other hand, can easily be created without missing any blocks. This chain length method removes the missing block penalty from the real life chains.

2.2.25 Preferred Fork Determination Procedure

When a party first joins the network, or rejoins the network after a hiatus, it may receive multiple forks from other participants and must choose the preferred fork to build blocks on. It uses the following procedure:

1. Determine the current slot based on the current time. Reject all future blocks in all forks.
2. If forks are unequal, select the longest fork. Note that the fork length is the number of slots spanned by the fork, not the number of blocks contained in it.
3. If the forks have any unshared epochs, first choose the preferred fork with unshared epochs (below) and reject other forks.
4. If there are still multiple forks, choose the fork with the highest approver stake stored at its head block. Use $A(\cdot)$ for tie-breaking if needed.

Determination of preferred fork with unshared epochs:

1. For each pair of forks, determine unshared epochs and blocks and then determine parties having created any block or approved any block or epoch within those unshared blocks in each fork. Note that epoch approvals with shared blocks are ignored (which may occur in a partially shared epoch). We call these the "signing" parties.
2. Determine the total stake fraction (of the total network stake) of all "signing" parties at the first separate block of each fork.
3. Select the fork with the larger signing stake fraction.

Figure 9 shows shared and unshared epochs of 2 forks. Epochs $e-1$ and e are shared while epoch $e+1$ is unshared.

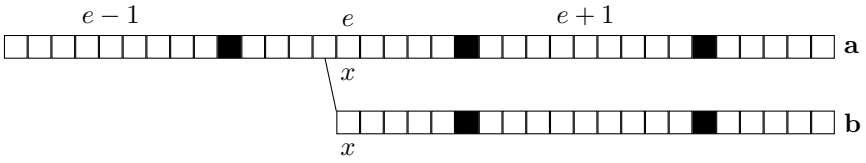

Figure 9: Shared and Unshared Epochs

Let us assume that there are l unshared blocks in a fork starting from block x. Let us denote the set of "signing" parties in a block r in a fork f to be $\mathbb{P}_{rf}$. The set of "signing" parties in fork f

$$\mathbb{P}_f = \bigcup_{r=x}^{x+l-1} \mathbb{P}_{rf} \tag{3}$$

Let us denote the stake fraction in slot x in a fork f owned by $\mathbb{P}_f$ parties to be S_{xf}. For each pair of forks, the fork with the larger S_{xf} is the preferred fork.

3 Analysis

3.1 Desiderata

In order to achieve a successful distributed ledger, a consensus protocol must provide the following properties [14][13].

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. for shipping sold goods. Finality, also called "strong consistency" [18], is durability comparable to the mainstream financial systems.

3.1.2 Chain Quality

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.

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

$$\epsilon \cdot T_r = \sum_{u=1}^{|\mathbb{U}|} s_u < (\rho - \delta - \nu_c - \nu_a - \nu_e) \cdot T_r \tag{4}$$

$$\rho > \epsilon + \delta + \nu_c + \nu_a + \nu_e \tag{5}$$

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

   While the corrupt fraction of stake ϵ is bounded, there is no bound placed on the number of corrupt nodes $|\mathbb{U}|$.

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 the adversary is not able to block 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. **Maximum Single Party Stake**. No single party or colluding parties hold a quorum stake in the network.

5. **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 10 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. Let us call a candidate block and its subsequent blocks, which may or may not persist, a candidate fork. Each of the candidate block and candidate forks will include the following stake changes:

1. Newly minted creator stake of $\nu_c \cdot T_r$ may go to different parties in different forks.

2. Newly minted block approval stake up to $\nu_a \cdot T_r$. The maximum variation among the forks is $(1 - \rho) \cdot \nu_a \cdot T_r$.

3. Newly minted epoch approval stake up to $\nu_e \cdot T_r$. This would be 0 for all blocks except for the last block in an epoch. The maximum variation among the forks is $(1 - \rho) \cdot \nu_e \cdot T_r$.

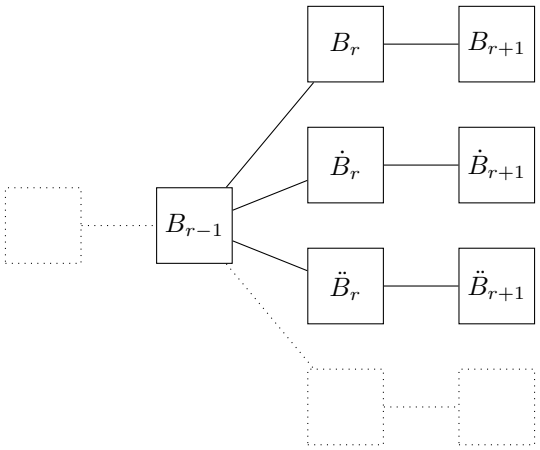

Figure 10: Common Prefix during Typical Execution

4. Transaction fees and transfer amount totaling up to a maximum of $\delta \cdot T_r$ may go to different parties in each fork. Let us also assume the worst case where the entire transfer stake is taken from the honest parties.

For the slot sl_r, honest parties as well as the corrupt parties will approve candidate blocks. Since adversary controls less than a quorum stake, at least some honest stake is required for a candidate block to be valid. Therefore, each candidate block must pass validation of at least one honest node.

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 fork in order for their candidate fork to persist. In each candidate fork at slot sl_{r+1}, the total stake is

$$(1 + \nu_c + \nu_a \cdot \rho) \cdot T_r \leq T_{r+1} \leq (1 + \nu_c + \nu_a + \nu_e) \cdot T_r \tag{6}$$

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 quorum stake in any candidate fork is

$$\rho \cdot T_{r+1} > \rho \cdot T_r \tag{8}$$

The stake not trusted to vote for the correct fork, denoted by A_{r+1}, is

$$\begin{aligned} A_{r+1} &= T_{r+1} - H_{r+1} \\ &\leq (1 + \nu_c + \nu_a + \nu_e) \cdot T_r - (1 - \epsilon - \delta) \cdot T_r \\ &= (\epsilon + \delta + \nu_c + \nu_a + \nu_e) \cdot T_r \end{aligned} \tag{9}$$

From Equation 5, quorum stake at any candidate fork

$$\begin{aligned} \rho \cdot T_{r+1} &> \rho \cdot T_r \\ &> (\epsilon + \delta + \nu_c + \nu_a + \nu_e) \cdot T_r \\ &\geq A_{r+1} \end{aligned} \tag{10}$$

Even if all corrupt parties approve conflicting 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 slot 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 be valid 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 (Adversarial Winning Blocks)**.** *In a typical execution, all winning blocks including those created by adversarial nodes are fully valid.*

Proof. Every candidate block needs approvals from a quorum stake to be placed in the chain. From proof of Theorem 3.1, the adversarial candidate block must pass validation of at least one honest party in order to win quorum approval. No honest party will approve an invalid block, therefore, all blocks placed in chain must be valid. □

Theorem 3.4 (Chain Quality)**.** *The blockchain maintained by honest parties is guaranteed to have some blocks contributed by honest parties.*

Proof. For each slot, a number of coin rolls are selected for block creation. Since the network maintains an honest stake majority, a majority of those coin rolls must belong to honest parties. Therefore, some blocks created by honest nodes will be inserted in the blockchain. □

3.3.3 Chain Growth

Theorem 3.5 (Chain Growth)**.** *In a typical execution the chain-growth property holds with parameters* $\tau \approx 1$*.*

Proof. At every round, multiple parties create candidate blocks and broadcast them to other parties. Due to a majority honest stake, a majority of those block creators would be honest and 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. □

3.4 Defense Against Attacks

3.4.1 History or Costless Simulation Attack

Theorem 3.6 (Costless Simulation Stake)**.** *For a Costless Simulation attack to succeed, the attacker must own private keys of nearly the entire stake at some point in history.*

Proof. Proof-of-Approval incorporates epoch approvals that are signed testimonials of the stakeholders for a chain. Since epoch approvals are non-competitive, provide large award and require little or no computation, all rational stakeholders would collect this reward and in process, sign their acceptance of the chain. Even if some parties are temporarily unable to participate due to network issues, the fork selection process looks at union of all epoch approvers between the competing forks. As a result, the "real" fork would have epoch approval from nearly the entire stake and an attacker can win only by exceeding it in their attack fork. □

Theorem 3.7 (Costless Simulation Cost)**.** *Acquiring private keys for a Costless Simulation attack on a Proof-of-Approval chain is likely to cost dramatically more than acquiring keys representing a simple majority in past.*

Proof. Stake distribution in a chain is likely to be Pareto distribution where the last few percentage of stake may involve nearly as many keys as the rest of the stake. This makes acquiring nearly all keys dramatically harder than acquiring keys of a majority of stake. In addition, the key holders would realize that the attacker has no choice but to buy every key they can find resulting in high prices even for keys that held tiny stakes in the network. □

3.4.2 Nothing at Stake Attack

Theorem 3.8 (Nothing at Stake)**.** *Proof-of-Approval does not suffer from Nothing at Stake attack since the approvers lose their award if they approve*

conflicting blocks.

Proof. Proof-of-Approval provides award for valid approvals, even on multiple blocks, as long as they are not in conflict, i.e., they share the same parent. An approver's award is maximized when they approves all non-conflicting blocks. But if they approves any conflicting blocks, the award vanishes. Therefore, the best strategy for a rational approver is to choose a parent block and approve all valid children blocks.

Parties are also rewarded for epoch approvals which also go away for approving conflicting epochs. As a result, parties are expected to approve a single chain in an epoch to maximize their rewards. □

3.4.3 Bribe Attack

Theorem 3.9 (Bribe Attack)**.** *In order for a Bribe attack to succeed, the attacker must own or collude with a stake larger than that specified by Equation 4.*

Proof. Theorem 3.1 shows that if the attacker stake is limited to that in Equation 4, forks larger than 1 block will not survive. Intuitively, for a block to be valid, it must have signed approvals from a quorum stake. Due to awards and transfer, an adversary can get blocks approved with slightly less stake than quorum but not lower than that specified by Equation 4. □

3.4.4 Stake Grinding Attack

Theorem 3.10. *Block creators cannot be manipulated.*

Proof. Block creator selection uses whole and fractional coins (excluding rolls) of the approver stake in the parent block. While the approximate approval stake may be predictable, the coins and the fractional coins are highly random and not predictable. If at some time, possibly due to poor network connectivity, the same group of parties approves successive blocks, it may be possible to guess the whole and fractional coins stake in the very near future but not for long.

A block creator, of the current block, has only one options to manipulate the block they are creating in order for them to be a block creator for the next block - willingly drop some approvals. Dropped approvals may change fractional coins so that the coin roll selection algorithm chooses one of the rolls they own.

Since the protocol always prefers higher approver stake, any dropped approvals will result in the created block to be less preferred than others. This is likely

to result in the attacker's current block not being placed in the chain. Therefore, such an attack does not succeed. □

Theorem 3.11. *Block creators are likely to change at every slot.*

Proof. The whole and fractional approval coins are highly random and will change dramatically from slot to slot. Even if the same set of parties approve the consecutive blocks, their fractional stakes would change at each slot due to approval award. In addition, slot id is added to the coin roll selection algorithm to ensure changes to the block creators. □

3.4.5 Stake Bleeding Attack

Theorem 3.12. *Proof-of-Approval does not suffer from Stake Bleeding attack.*

Proof. The long range fork selection algorithm compares stakes only at the first unshared block. Therefore, this algorithm does not suffer from stake bleeding attack. □

4 Practical Implications

4.1 Award Competition Self Selection

The protocol rewards three separate activities - block creation, block approval and epoch approval. The first two activities benefit from faster computation and better network connectivity. Parties choosing to win these rewards would likely use a cloud server instance. A small cloud instance is likely to suffice which can be acquired for approximately $5/month.

The third activity, epoch approval, offers awards as large as the block approval, but requires little computation or network performance. We expect most, perhaps all, parties would participate in this award. Since this award is not a competition, each and every party with any stake that tries would win this award.

4.2 Additional Storage Requirements

The protocol presented here uses more storage than many other protocols because it stores approvals 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.

4.3 Limited Number of Block Creators

The number of block creators must be limited to have reviewers a reasonable chance at reviewing them. Moreover, an excessive number of candidate blocks may result in communications being bogged down. Therefore, only a small fraction of parties are allowed to create block in each slot. But this number is large enough to have at least one acceptable block in each slot most of the time.

4.4 Slot Period

Slot period should be kept small enough to let nodes having better network connections win in block creation and block approval. Keeping slot smaller provides an incentive for nodes to strive for faster and more reliable network connections resulting in higher liveness of the blockchain.

Reducing the slot period achieves another desired property – higher transaction throughput of the blockchain.

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] A. Back, *Hashcash - A Denial of Service Counter-Measure*, `http://www.hashcash.org/hashcash.pdf`, 2002

[5] A. Poelstra, *On Stake and Consensus*, `https://download.wpsoftware.net/bitcoin/pos.pdf`, 2015

[6] A. Poelstra, *A Treatise on Altcoins*, `https://download.wpsoftware.net/bitcoin/alts.pdf`, 2016

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

[8] W. Li, S. Andreina, J. Bohli et al., *Securing Proof-of-Stake Blockchain Protocols*, Data Privacy Management, Cryptocurrencies and Blockchain Technology, 2017

[9] Ethereum, *Proof of Stake FAQ*, `https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ`

[10] BitFury Group, *Proof of Stake versus Proof of Work*, `http://bitfury.com/content/downloads/pos-vs-pow-1.0.2.pdf`, 2015

[11] P. Gazi, A. Kiayias, A. Russell, *Stake-Bleeding Attacks on Proof-of-Stake Blockchains*, Cryptology ePrint Archive 2018/248, 2018

[12] D. Larimer, *Transactions as proof-of-stake*, `https://bravenewcoin.com/assets/Uploads/TransactionsAsProofOfStake10.pdf`, 2013

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

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

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

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

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

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