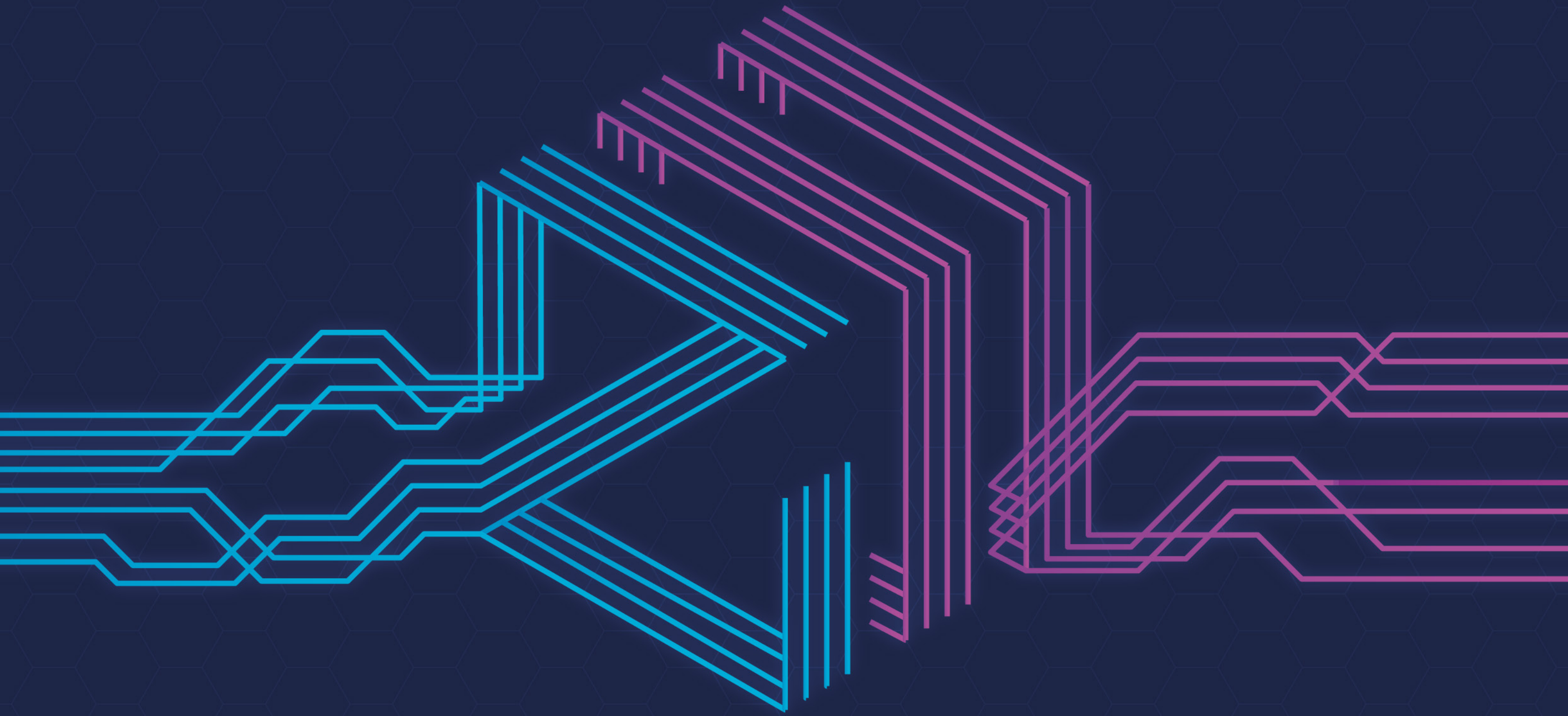
ZooBC White Paper

New Strategies in Blockchain



by Barton Johnston and Roberto Capodieci, with the help of Stefano Griggio

Release: V1.1 P - 2 April 2020

Abstract

Since the unfolding of Bitcoin, a plethora of blockchain technologies has emerged, each approaching certain limitations of the original design. In the same spirit, in the first of what we intend to be a sequence of gradual improvements, in this paper we propose a new blockchain architecture and generally describe its structure. Among other aspects, we focus on a more evenly distributed block creation, a fair distribution of the network rewards among the nodes that do useful work for the network, a reduced blockchain download time that, no matter how bloated the blockchain is, remains a constant, and to create a flexible technological base for others to implement on top of it the business logic specific to their domain. To start we explore the security and performance implications of each new blockchain, and where those differ from ZooBC, our blockchain technology.

Disclaimer

Online Live White Paper

If you are reading this document in a PDF file or a hardcopy, and want to access the online version to help with some comments and edits, please use the QR code, or click **here.** For the older versions of the ZooBC White Paper, **please see Appendix 1 at Page 112.**

Latest Release

Download (and print, if you need to), the latest version of this white paper (at the moment V1.1) in PDF format at this link:

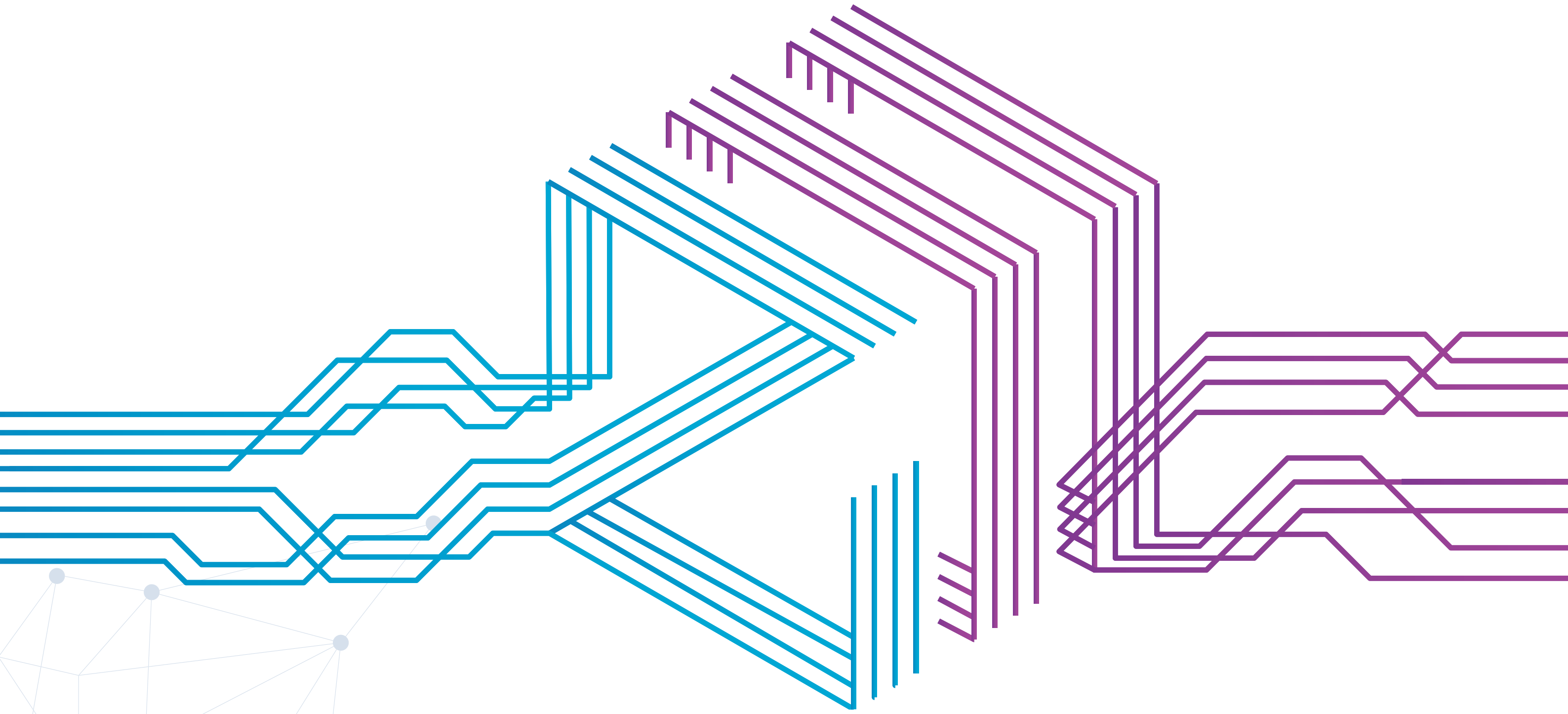
https://bcz.bz/ZooBC-WP

Share this White Paper

NOTE: *The Proof of existence of this document has been done only AFTER the document is completed. The Proof of existence URL of this document can be reached through this short URL:* ***https://bcz.bz/ZBCWP1-1-PoE***

Privacy Considerations

Before looking at the structure of ZooBC V1, we should address an important point. Moving into the future of blockchain and decentralized systems, there is increasing concern over the entirely-public nature of the data on the blockchain. While privacy in ZooBC is simply handled with data encryption, other forms of privacy management, such as zero proof transactions, will be implemented in ZooBC V2, and they stand out clearly in our minds as important to address in the technology as a whole. **In Appendix 2, we list the thoughts that are guiding us as we begin designing V2 of the ZooBC blockchain technology.**

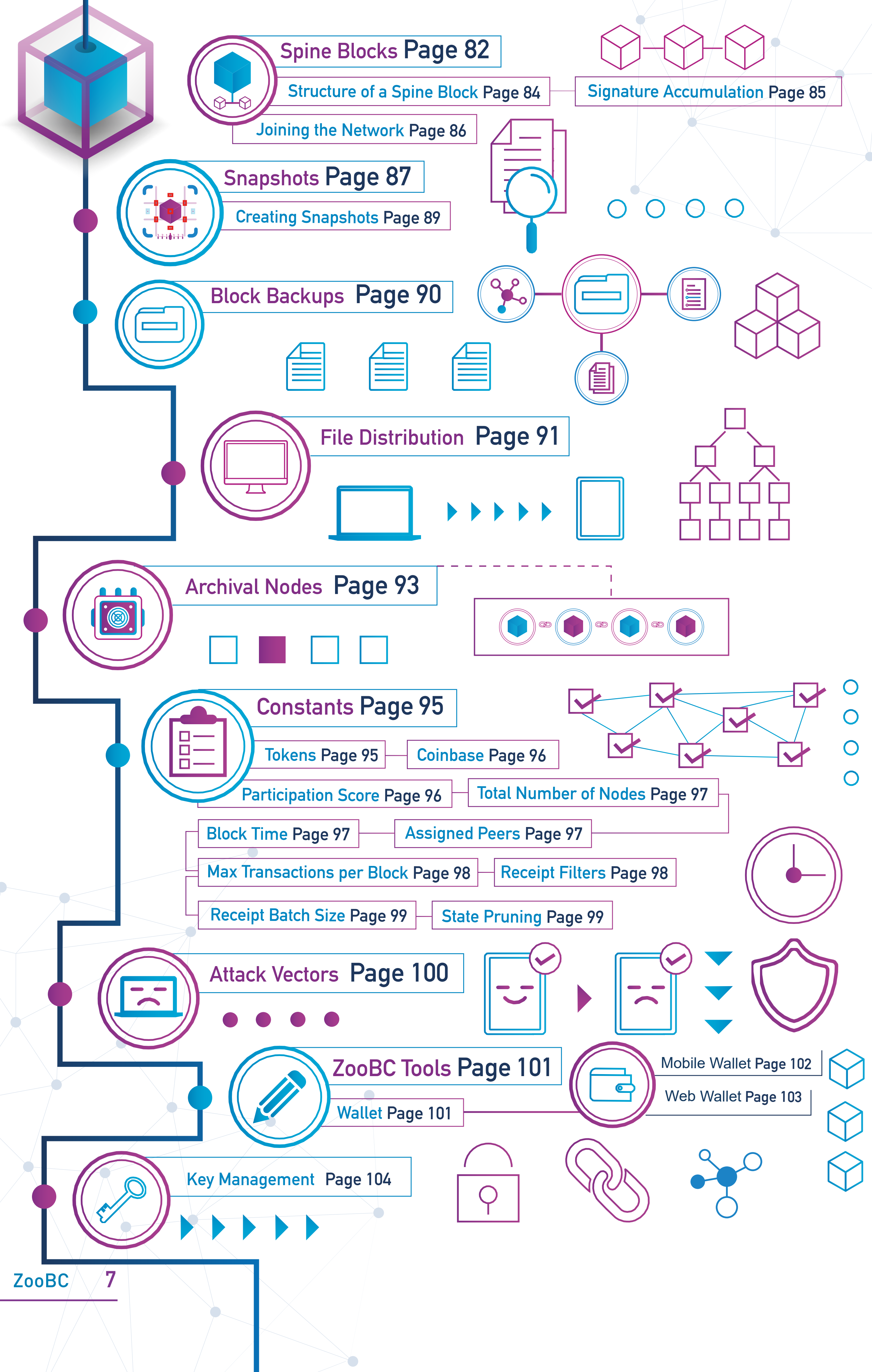


White Paper Index

for a normally formatted index, see page 147

White Paper Index

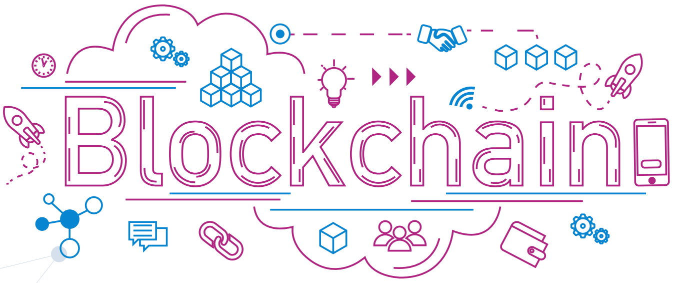
White Paper Index



Introduction

As blockchains have grown from Bitcoin into a multitude of diverse technologies, a few key problems have remained largely unsolved. Many technologies take many approaches to ameliorate some of the issues, but some remain unaddressed, and many others remain unsatisfactorily solved. In the first version of our proposed technology, we do not aim to address all of them, but address some, and lay a foundation to address others. A few such issues are these:

Consensus algorithms are still evolving rapidly. Broadly, consensus algorithms of existing blockchains fall into three categories: ***scarcity of external resources*** (proof of work, **scarcity of internal resources (such as proof of stake), and federated consensus models.** The first two derive their security from who has the most money, which is a weak security model for a blockchain where the objects of interest are non-financial; and most federated models make security compromises around how members are elected to or ejected from the federation.



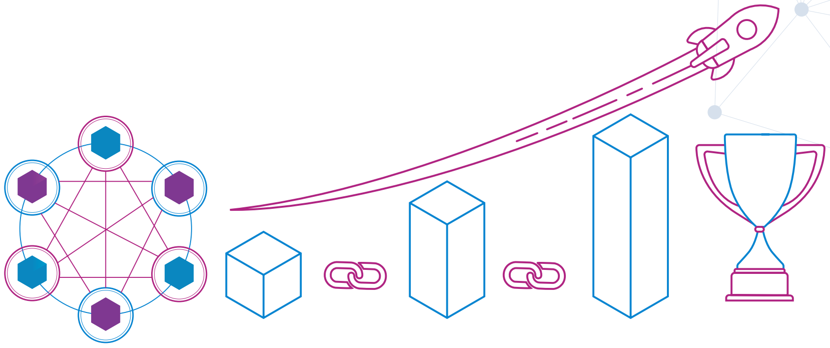
Blockchains typically do not reward the nodes which do meaningful work for the network. Most chains reward only those nodes which create the blocks, but do not have a method to measure which nodes are gossiping messages and thereby keeping the network decentralized. This is largely because it is not easy to achieve consensus on the behavior of so many nodes in a trustless decentralized way, however the obvious problem remains: a perfectly efficient actor will learn to create blocks but not gossip transactions.

Blockchains download slowly, and the length of the download depends on the age and activity level of the chain. In many cases such as Bitcoin or Ethereum, the chain becomes so large that it is a large investment or undertaking to even join the network and operate a node. The difficulty for a new node to join the network makes it difficult for networks to grow large enough to have the true robustness of decentralization.

Decentralized applications running on blockchains are currently restricted to either being strongly-coupled, such as smart contracts which may only be coded in the blockchain's language in a limited environment, or weakly-coupled, such as external pseudo-centralized applications which read and write important data to the blockchain as a storage layer. There is a lot of room between these extremes to be explored, to allow the creation of decentralized applications which leverage the decentralized network of the blockchain and its security properties, while not being bound entirely into a sandbox where each instance merely replicates the results of the others.

The ZooBC Project as a whole aims to analyze, experiment, and ultimately develop solutions to these and other problems facing the ecosystem of blockchain technologies.

Project Goals



Our long-term vision with the **ZooBC** project is to develop a public, scalable, decentralized application platform and also to create a core blockchain technology that can be easily adapted to other business use cases for private deployments. With ZooBC V2.0, which will be the subject of a later whitepaper, Blockchain Zoo is developing an approach towards decentralized applications ("DApps") which is radically different from the conventional "smart contracts" approach used by Ethereum and other blockchain technologies. However, before working towards this goal, we have elected to start with ZooBC V1.0, a more conservative technology that lays the foundation for future work.

This first version is similar to other existing blockchain technologies but with a few extra key core mechanics which will be leveraged more thoroughly as ZooBC progresses into future versions. These core functions bring some unique value to the first version of ZooBC but, will deliver vastly more as new functionalities, such as the ***DApp platform, geo scalability***, and ***infra-chain transactions***, are implemented (more on this in the V2 of the white paper.)

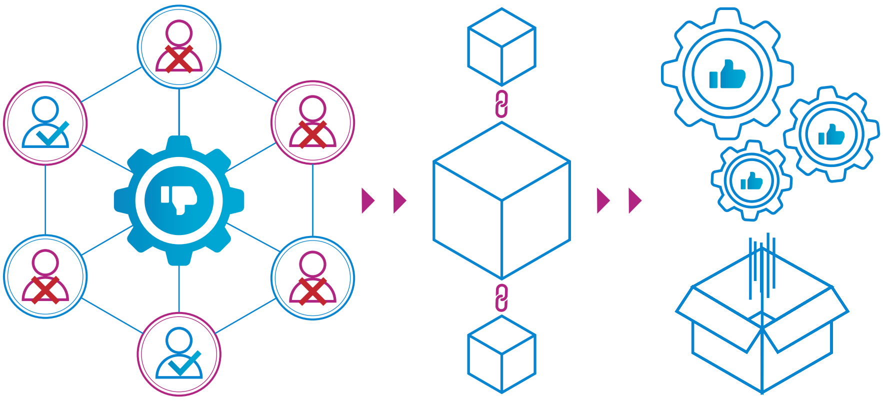
The first major diversion from previous blockchain technologies is a new consensus algorithm called ***Proof of Participation*** that has been architected, engineered, and implemented by Blockchain Zoo. In the context of the DApp platform that will be implemented in ZooBC V2.0, the Proof of Participation algorithm gives ZooBC a large consensus-tracked set of participating nodes with incentives to remain online, which can later be organized into sub-federations that may perform more efficient consensus algorithms for DApps. In the following pages we develop the motivation for pursuing a new kind of algorithm, and how its design logically follows from the development of previous popular consensus algorithms.

The second major diversion from previous blockchains is the use of a lightweight cryptographic shortcut made of special blocks created once per day, and called ***"spine block"***. This secures the state of the ZooBC blockchain at certain checkpoints and allows new nodes to identify and download the latest state of the network, cryptographically secured, without having to download and apply all the previous blocks and transactions. In the context of a DApp platform, this checkpointing system can be used to secure the state of many apps, or even many blockchains, in one single place. For this first version of ZooBC, its utility is limited to enable new nodes to safely and quickly join the network at the current state.

The third major diversion from previous blockchains allows a user to choose which digital signature algorithm will secure his account, and to configure an account so that it requires approval to receive transactions. These changes are targeted at increasing the blockchain's compatibility with various government regulations, for example using digital signatures recognized in court to validate blockchain transactions and proving a user agreed to receive certain funds or assets before they are attributed to his ownership.

This paper will discuss in more detail the first release of the ZooBC technology (ZooBC V1.0 and enanchements toward the V2.0). **For a better understanding of the blockchain terminology used in this paper, please refer to the blockchain glossary in Appendix 3 at Page 116.**

Why Does Blockchain Technology Need Another Consensus Algorithm?

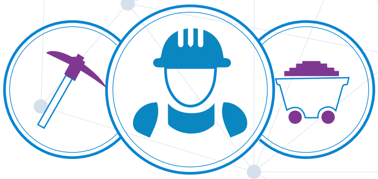


To understand why we have opted to implement novel mechanics to secure our blockchain, this section offers an overview of various existing consensus algorithms. This is for those readers wanting a more detailed account of the history of consensus algorithms and the reasons why we thought it necessary to develop beyond existing work. For those wanting to go straight to ZooBC's specifications and technical details, only the "Proof of Participation" paragraph is relevant.

The blockchain space is saturated with attempts to improve efficiency, security and fairness in the way that nodes reach a consensus on the history of events witnessed by the network. While the explosion of strategies may seem overwhelming or unnecessary, each project (some more than others) is doing its part in exploring the properties and tradeoffs yielded by each approach, and the crypto community is collectively narrowing down the proposed consensus strategies darwinistically until only the strongest are left standing.

Here a brief overview of the major approaches to blockchain consensus, and our reasoning to claim Blockchain Zoo's Proof of Participation as an improvement over its predecessors. **For a more detailed overview and visualization, please see Appendix 4 at Page 129.**

Proof of Work Consensus



The Bitcoin white paper introduced the concept of using accumulated "Proof of Work" as a method for any node to agree on which blockchain, among forks, should be trusted. This approach was very powerful because it allowed nodes to independently and objectively agree on one proposed history of events among many alternatives, in a way that resists a ***"Sybil attack"*** (because votes are counted by CPU cycles, not by accounts.) While many insist that Proof of Work is still the safest way to secure a blockchain, time has shown some undesirable properties of the algorithm such as the increasing energy usage, the centralization of the mining power, and potential for externalized control. **Find out more about PoW in Appendix 4 at Page 129.**

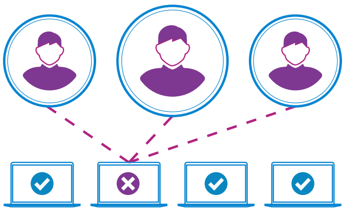
Proof of Stake Consensus

These concernsmotivated some to develop an alternate consensus algorithm to objectively choose between proposed versions of the blockchain history called "Proof of Stake". In this approach, the likelihood of a network participant to add a block to the history is computed according to how many tokens on the network she possesses, and the block she creates is proven to originate from her via a digital signature. In this way, which chain required "more work" to create is simulated by a calculation of which nodes added blocks at which times and their relative stakes. This design requires minimal energy and guarantees that, in a fork, the nodes will choose the blockchain created by the majority of highly-invested network participants -- in other words, those who have a larger stake of tokens locked to create new blocks. However, this strategy still has some undesirable properties such as a majority of the tokens being in the hands of only a few participants, the possible creation of the alternate blockchain history controlled by only a small number of private keys. **Find out more about PoS in Appendix 4 at Page 129.**

Federated Consensus

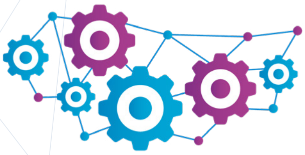
These consensus algorithms have been well-studied long before the emergence of blockchain technology for use in other distributed systems. While we feel such strategies effectively address the concerns above, in other ways they are a step backward from Proof of Work and Proof of Stake. Federated networks are no longer "permissionless", and the set of federated entities is usually well known so we feel a pure federation is not acceptable for secure decentralized consensus. **Find out more about the Federated Consensus in Appendix 4 at Page 129.**

Delegated Proof of Stake Consensus



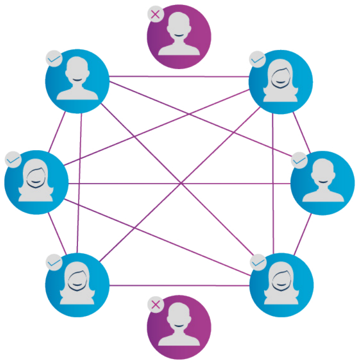
One of the most popular modern approaches to improving the scale of a blockchain network is to use ***"Delegated Proof of Stake"*** Consensus, where the accounts of the blockchain vote, with, their stake, a small number of nodes running large enough hardware, to become block creators and thus support a high transaction volume blockchain. While this approach can dramatically increase the throughput of the network, it does so at the expense of decentralization, having similar flaws as conventional Proof of Stake and small Federations. As described above, specifically, the ease of quickly collecting enough stake to control the network, and the ability of a small number of block creators to conspire to censor transactions. **Find out more about the DPoS in Appendix 4 at Page 129.**

Byzantine Fault Tolerant Consensus



Another popular strategy for increasing the transaction throughput of a decentralized network is an algorithm called ***"Practical Byzantine Fault Tolerance"***. This algorithm is especially used in Federated consensus, where the participants are pre-selected, because it carries a particular weakness in the face of Sybil attacks (when one attacker can operate many nodes on the network) which would make it unsuitable for some pBFT networks. **Find out more about the pBFT in Appendix 4 at Page 129.**

Proof of Participation Consensus



Based on consideration of the various flaws and tradeoffs in the consensus mechanisms explored above, ZooBC adopts a few elements of Proof of Stake and of Federated consensus strategies, combined with a novel algorithm developed by Blockchain Zoo to prove that a node is performing useful work for the network. We call this **"Proof of Participation"** consensus. ZooBC maintains a federation of nodes that we call the "Node Registry". Only nodes within the registry are permitted to create blocks, and their probability to create the next block is more or less equal. This is similar to Federated Consensus. However, any node operator can apply for a spot in this registry, and their admittance into the registry is governed entirely by the protocol rules, not by any centralized entity. The rate at which new nodes are added to the registry is strictly limited by the protocol, and the selection of which applicants will be added is governed by protocol rules that can be set based on the use case of ZooBC deployment. For example it can be based on participation efforts, score gained by nodes based on a specific rule, or by mere random selection. For ZooBC public open blockchain, to give value to the core token, we have decided to govern the selection of what nodes can stay in the registry, by how much stake they are willing to lock while they are in the registry. As nodes queue to enter the node registry, priority is given to nodes with a higher locked stake. This method uses a concept of Proof of Stake, to the extent that staking tokens (a scarce resource on the network) is used as a Sybil prevention mechanism, essential for a new blockchain. **Find out more about the PoP in Appendix 4 at Page 129.**

With this explanation of why a new consensus mechanism is needed, the white paper continues, in the next section, with a general description of the technology used in ZooBC.

ZooBC: a PoP Decentralized Application Platform

This section of the paper presents a reasonably complete explanation of the system in technical detail. A future discussion of security considerations and attack vectors assumes the reader's familiarity with the following mechanisms, which enables us to reason about how they combine to form the security properties of the system as a whole. This will be further developed in future versions of the white paper

Interested in contributing to the code? Request access to GitHub repository **here**

VIDEO

Watch ZooBC ALPHA videos

Watch ZooBC explainer videos

FORUM

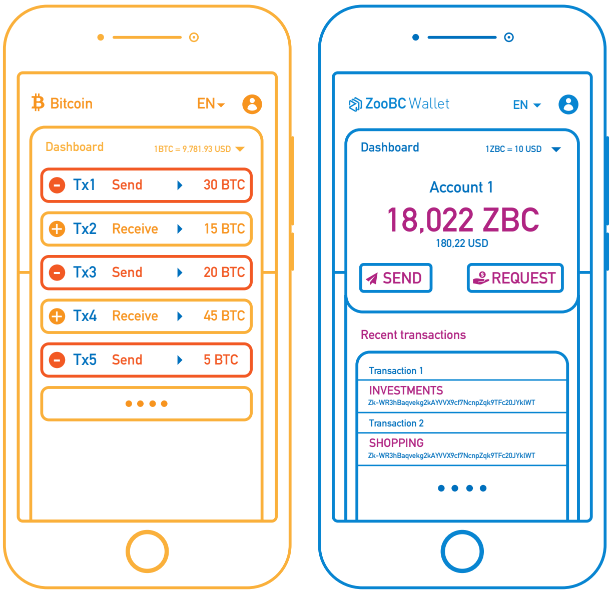
Join discusions about ZooBC

ZOOBC Q&A

Get your answers on ZooBC questions

Accounts

The very first blockchain (Bitcoin) uses an "unspent transaction output model" of a node's current state. In this model, there is a large pool of "transaction outputs" which can be "spent" by anyone possessing the private keys corresponding to the addresses identified as the receivers of the transaction. With this model, to compute the balance of an account, a node needs to apply in sequence the first transaction through the last. While this model confers certain advantages, it is limited to modeling the ownership of titles or assets.



🟧 Bitcoin Model 🟦 ZooBC Model

Subsequent technologies such as Ethereum adopted a more general "account model", where the state of the database is composed of "accounts" and their properties at any given point in time. For the same reason, in ZooBC we also use an account model to store the state of balances and other properties belonging to users in the system. With this model, the balance of an account and other properties can simply be queried from the database.

ZOOBC Q&A

Ask questions about Accounts

Account Addresses

Account 1

bc1qar0srrr7xfkvy5l643lydnw9re59gtzzwf5mdq

Most blockchain software picks a specific digital signature algorithm and requires all users to create key pairs using that algorithm. For our protocol we take into consideration two key factors: that some signature algorithms are increasingly respected in some countries by courts of law (government-issued digital signatures and eID cards), as well as the convenience of sending funds to an account for which you are already certain the other party possesses the private key, although on another blockchain.

Therefore we design our protocol so that a user can select his own signature algorithm from a set of supported **address types.** Each type specifies a unique address type code, as well as the format of the address, and the format of the corresponding digital signature. In this way it is possible, for example, to send funds to your friend's Bitcoin address on the ZooBC blockchain, knowing that your friend will then be able to use his Bitcoin private keys to sign valid spends of those funds on our blockchain. To be clear, this allows the use of a Bitcoin address in ZooBC blockchain, but doesn't mean that funds are transferred between the two blockchains. This will be possible with a dedicated DApp in the future versions of ZooBC.

VIDEO

What address types does ZooBC support?

Account Properties

Each account can have a set of ***properties*** assigned to it by itself or other accounts. In some cases the names and values of these properties may be arbitrary, while in other cases (especially in customized versions of the blockchain) certain property names may be given particular rules for how they are set and how they affect the consensus logic.

FORUM

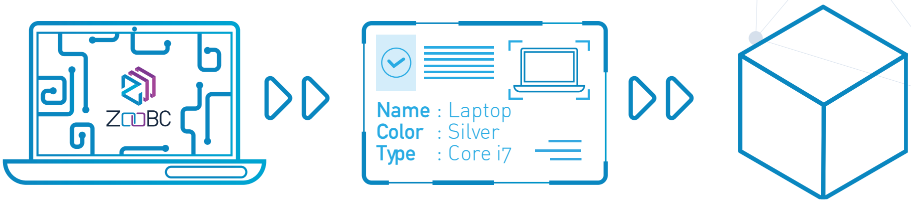
Join discussions about Account Properties

Account Types

While the most common type of account is for users, there may be other account types which share some characteristics of accounts (such as having a blockchain address, being able to have properties assigned to it, etc), but which have other characteristics which may be customized in the consensus rules.

One such account type is an ***asset account,*** which represents a single non-fungible asset which grants one party the right to modify its properties, and allows its ownership to be transferred from account to account. It could be used, for example, for warehouse receipts, land ownership titles, etc.

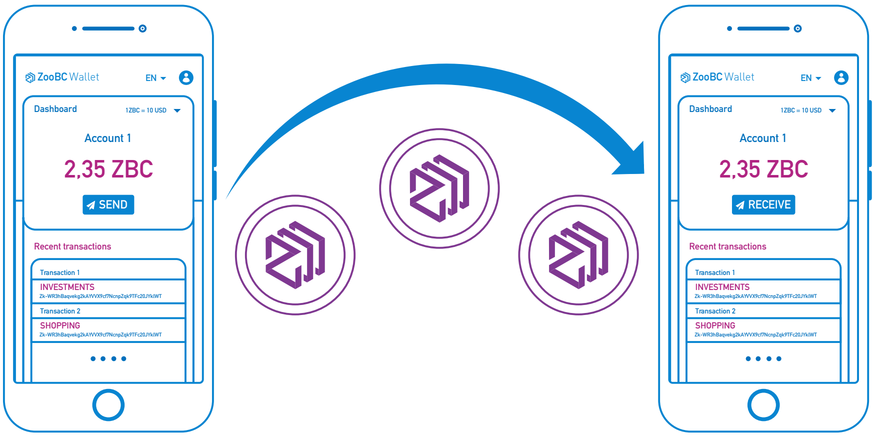
Digital Twins



The Digital Twins core principle is, for a physical entity or an asset, a digital equivalent exists in the blockchain. To replicate a physical entity – be it a machine, infrastructure or a living being – data is extremely important. The nature of data, consisting of physical attributes, inter - object interactions and future states, will be seamlessly exchanged between the digital and the physical worlds using blockchain. ZooBC allows users to create special account types to represent assets and be the digital part of the twins.

Transactions

Every action executed on the ZooBC blockchain by a user is encoded in a ***transaction*** that specifies the action the nodes should take, the action parameters, a data payload, and have all this digitally signed using the sender's private key.



A transaction may be as simple as transferring tokens from one account to another, but could have unlimited complexity, so long as its core application logic satisfies the properties that both the transaction validation and its execution are purely deterministic operations which only read from, and write to, the portion of each node's database managed by the consensus algorithm.

FORUM

Join the discussion about Transactions

ZOOBC Q&A

Ask questions about Transactions

Transaction Types

We do not support, nor plan to support, general ***"smart contracts" (on-chain code)***, despite the popularity of this approach, because we find it unsuitable for serious decentralized business logic. Code that cannot be updated in case of bugs or changing business rules (even if by a democratic process of distributing a node software update) is a liability. Further, implementing a VM (Virtual Machine - an isolated virtual computer inside a computer) that supports arbitrary logic, increases both the blockchain code complexity, and the user complexity in terms of computing transaction execution costs and interface design.

While in future releases of ZooBC it will be possible to deploy ***decentralized applications***, the first version will allow integration with centralized services on the base transaction, and is deployed with a set of ***transaction types*** recognized by the network, where each type defines a particular behavior. While the initial set of transaction types we include is limited, our reference implementation is structured in such a way that an organization producing a customized version of this blockchain can easily add or modify the transaction logic to their node software and corresponding wallet software.

FORUM

Join the discussion about Transaction Types

Transaction Propagation

When a transaction is first received from a peer or from a wallet software, the node will first confirm that the transaction is legal. This includes validating that the transaction is closed by a valid digital signature for its sending account address and that the parameters of the transaction are valid according to the transaction type's rules and the current state of the database (such as having enough balance to send funds).

If the transaction received is valid, the node will propagate the transaction by transmitting it to other connected peers, as well as store the transaction in its local mempool. In this way, valid transactions are echoed across the peer-to-peer network until they are retained in the mempool of the majority of nodes.

Upon receiving a valid transaction, each node will also return a special object we call a receipt to the sender. Receipts are used in the Proof of Participation algorithm described below.

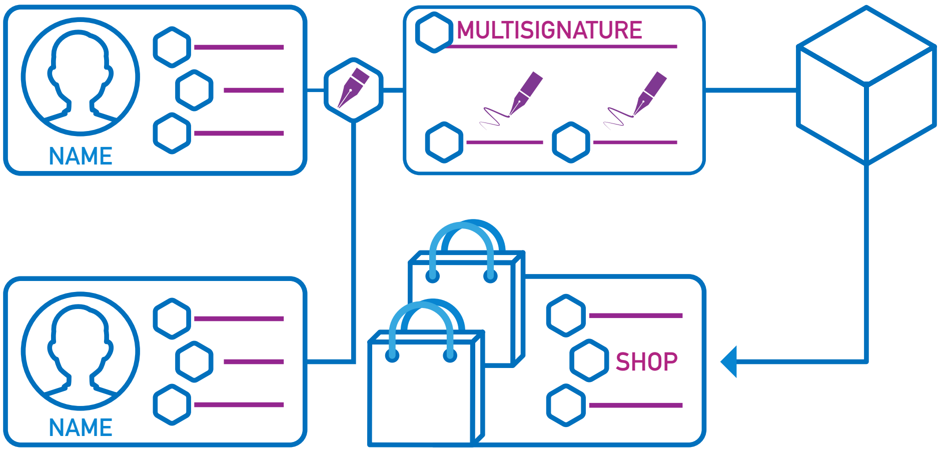
Transaction Application

When a node receives and validates a new block (described in the section on "blocks" below), it will contain an ordered list of zero or more transactions, which the node will then apply in sequence.

For a node, applying a transaction means executing the rules associated with the transaction's type to update its local database from an old state to a new state. In the simplest example, this can mean deducting the balance from one account and adding it to another.

Multisig Transactions

Multisig accounts are accounts that require multiple signatures to post valid transactions. This is needed when at least X out of Y people need to agree for a transaction to be executed. ZooBC supports a special type of multisig account, which may specify a set of cosigners who may approve transactions from that account. An action by a multisig account is managed by the cosigners by submitting a sequence of transactions which adds to the necessary amount of signatures until enough signatures are reached for the action to be executed. The mechanics of multisig accounts and multisig transactions are described in more detail below in the Multisig section.



Transactions Attachments

In future versions of ZooBC, transactions will be allowed to specify large attachments to be stored in a distributed file system by the network. While the transaction itself must be propagated to all nodes, the attachment may only need to be propagated to a few nodes responsible for storing the file for others to download. More details on this are given in the section below on File Distribution.

Escrowed Transactions

When two parties must swap goods online, it is often useful to have a trusted third party keep the goods in **escrow,** such that the swap is only executed when both parties have committed their assets. Traditionally, this trusted third party holds both of the assets, and if they are not in fact so trustworthy, they may abscond with both. Therefore it is useful to have a system in which the trusted third party may only approve or reject the transfers between parties, but in no case becomes the owner of the assets.

To facilitate this use case, ZooBC users may optionally include in the transaction the account address of an **approver,** which may either accept or reject the application of the transaction. Transactions which require approval are kept in an "escrowed" state by the blockchain, such that the funds or other assets they confer ownership of cannot be used by either the sender or the receiver until the explicit approval or rejection is completed, or the timeout returns control of the assets to the sender.

The transaction may also specify:

A a custom timeout: the number of blocks until, or the DateTime at which, the transaction is automatically rejected (by default 1 day). The maximum timeout is 1 month

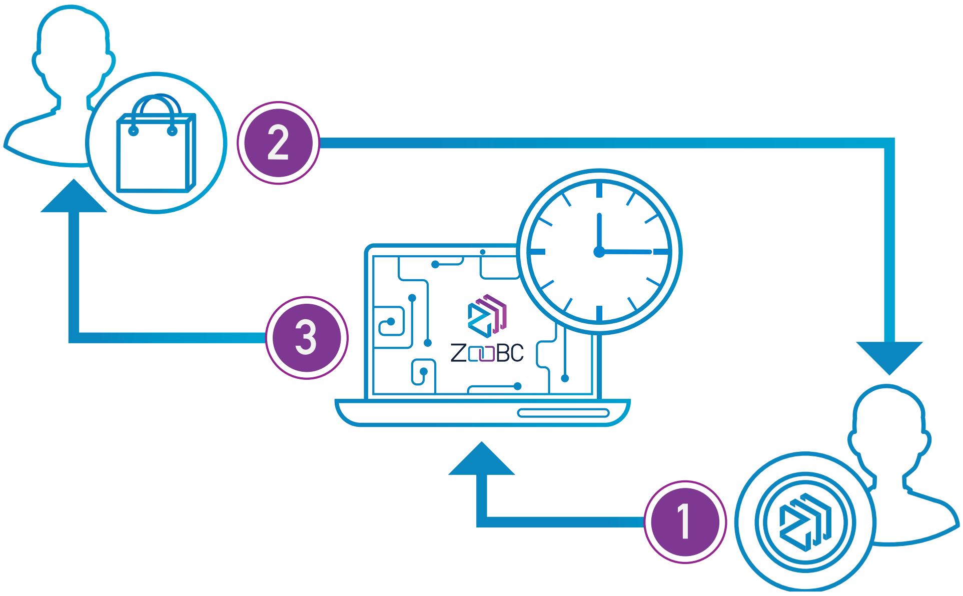
B a commission: an amount of tokens which will be paid to the approver if he accepts / rejects the transaction before the timeout, by default, the cost of a transaction

C instructions for the approver: in binary or a JSON format if the approver is an application on a server, or in a human language if the approver is a regular user of ZooBC

D the behavior for when the transaction times out: automatically approve or reject

In the case that the timeout is reached before the approver sends an approval or rejection, the transaction is automatically approved or rejected and the commission is returned to the sender.

In order to approve or reject an escrowed transaction, the specified approver must broadcast an ***approval transaction,*** referencing the hash of the pending escrowed transaction, and specifying whether he approves or rejects it. While the approval transaction can be submitted automatically by centralized services, when a user needs to manually submit the approval for a transaction, the technical complexities are hidden by the wallet UI.



One use case of such escrow mechanism would be to configure a centralized server which manages an approver account, and which is programmed to automatically approve transactions only when certain conditions have been met. For example, an escrowed transaction may specify that it should only be approved once 10 Ethereum have been transferred to a particular ETH account, and set the approver to be a server which will monitor the Ethereum network to determine whether to release the funds.

Alternatively, any account may be configured to require explicit approval to accept any transaction addressed to it. In this case, if an approver account is not specified, the recipient account automatically becomes the approver. The rest of the mechanics defined above apply in exactly the same way, with the receiver of the transaction functioning as the approver.

To achieve the current normal behavior of blockchain transactions, any transaction which does not specify an approver account, and for which the receiver is not configured to require approval to accept transactions, is applied immediately upon its acceptance in a block.

The owner of an account may at any time enable or disable this mandatory approval behavior on the account by broadcasting the desired account property setting with a ***required approval transaction*** and specifying whether mandatory approval is enabled (default is “no”).

Liquid Transactions

In some cases, it may be useful to represent a continuing stream of regular payments from one account to another for the duration that a service is being used. Conventional transactions are a poor fit for this purpose because they must be explicitly created and signed by a user's wallet each time. Therefore, ZooBC implements ***liquid transactions.***

A liquid transaction specifies an amount of funds to be paid and the duration over which those funds should be paid. During this time, the funds will slowly and continuously move from the sender's account to the receiver's account. The sender may cancel the ongoing liquid transaction at any time, only having paid the amount that the seller has already received. And the receiver may access the funds as soon as they are received in his account by the passage of enough time.

External applications may easily reference an ongoing liquid transaction to decide whether to grant a user in their system (corresponding to the sender's ZooBC account) access to features or services on their platform. This feature could also be used for payment of salaries or allowances on a continuous basis.

Transaction Fees

Each transaction must include a ***fee*** in tokens to pay for its execution on the network. The fee for each transaction, together with new tokens generated at each new block, are distributed among the nodes in the "node registry". Requiring a fee for each transaction serves as a form of spam protection, ensuring it is costly to overwhelm the network's limited transaction volume. It also serves to incentivize block creators to include as many transactions as possible, maximizing the collective reward for producing blocks.

The ***minimum fee*** for a transaction is computed differently for each transaction type, then multiplied by a fee scaling constant which can be periodically adjusted. A user posting a transaction may pay more than the required minimum fee in order to be accepted in a block faster when the network load is high, but a transaction with a fee lower than the computed minimum will be rejected. For transaction A, this minimum fee can be computed:

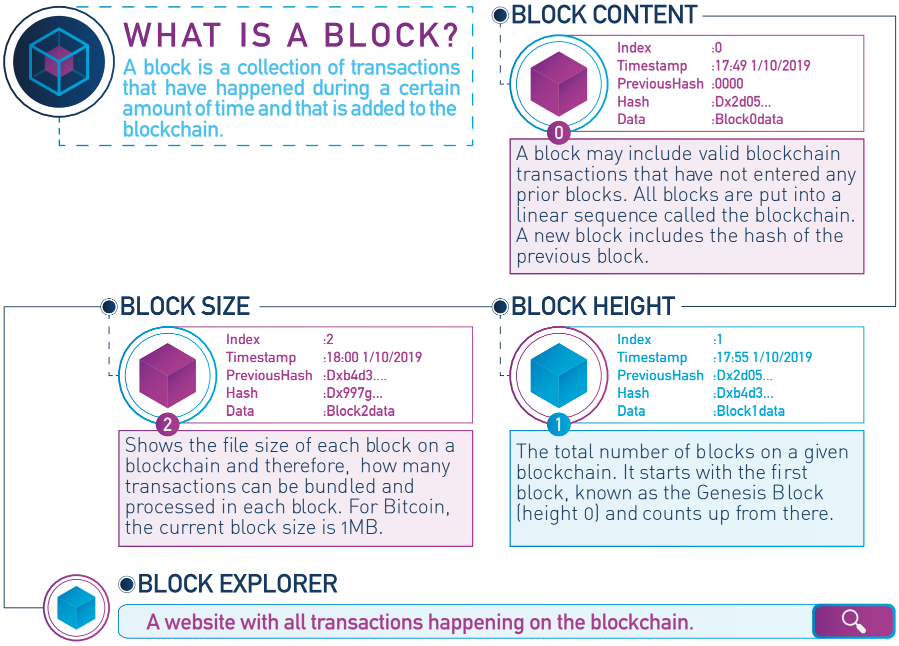
$$\boxed{\text{min\_fee}(\, Tx_A \,) = \text{type\_fee}(\, TxType_A,\ Tx_A \,) * \text{fee\_scale}}$$

ZOOBC Q&A

Ask questions about Transaction Fees

Blocks

Unlike a centralized system that receives requests from the clients in a uniquely ordered fashion (serialized), in a decentralized system such as a Peer to Peer network (where users can use any node as an entry point to post data to the network) each node may receive transactions in a different order.



WHAT IS A BLOCK?

A block is a collection of transactions that have happened during a certain amount of time and that is added to the blockchain.

BLOCK CONTENT

Index	:0
Timestamp	:17:49 1/10/2019
PreviousHash	:0000
Hash	:Dx2d05...
Data	:Block0data

A block may include valid blockchain transactions that have not entered any prior blocks. All blocks are put into a linear sequence called the blockchain. A new block includes the hash of the previous block.

BLOCK SIZE

Index	:2
Timestamp	:18:00 1/10/2019
PreviousHash	:Dxb4d3....
Hash	:Dx997g...
Data	:Block2data

Shows the file size of each block on a blockchain and therefore, how many transactions can be bundled and processed in each block. For Bitcoin, the current block size is 1MB.

BLOCK HEIGHT

Index	:1
Timestamp	:17:55 1/10/2019
PreviousHash	:Dx2d05...
Hash	:Dxb4d3...
Data	:Block1data

The total number of blocks on a given blockchain. It starts with the first block, known as the Genesis Block (height 0) and counts up from there.

BLOCK EXPLORER

A website with all transactions happening on the blockchain.

A blockchain system fundamentally solves the problem of uniquely ordering all transactions which have been broadcast to the network. Nodes are chosen in a pseudo-random lottery to append sets of transactions to the network's total transaction history. We call such a set of transactions, along with some meta-information, a block. As indicated by the name "blockchain", *blocks* are cryptographically chained together by including in the metadata of each block the hash of the previous block. Consequently, to modify the contents of any previous block would require all subsequent blocks to be re-created.

VIDEO

Why does ZooBC blockchain network need two types of blocks?

FORUM

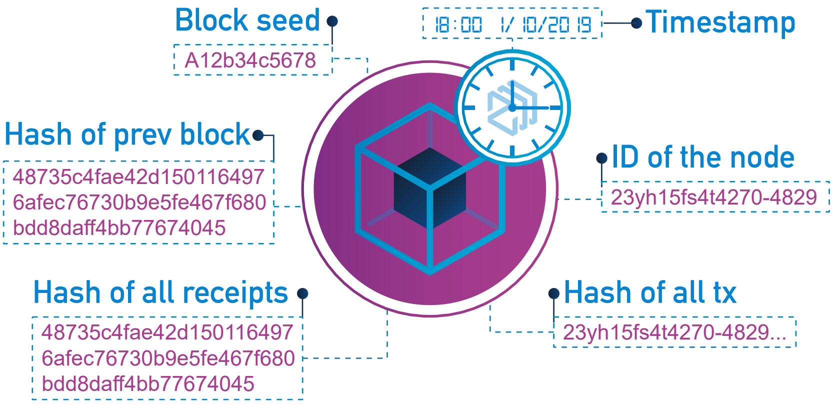
Join the discussion about Block Backups

ZOOBC Q&A

Ask questions about Blocks

Structure of a Block

Each block contains several key pieces of information, which become effectively immutable as more blocks are later chained on top of it. The block contains the timestamp at which it was created, the ID number of the node which created it in the registry, a hash of all transactions included in the block, a hash of all receipts included in the block (described in the Proof of Participation section), the hash of the previous block, a special parameter used for synchronized random number generation we call the "block seed", and a few other properties.



When a node receives a new block over the network, a series of validations will be performed over the block data itself, any transactions referenced by the block, and any receipts included in the block, such as ensuring that the previous block hash matches the actual last block hash the node has seen, that adequate time has passed since the timestamp of the previous block for the new block creator's position in the priority list of next creators, that the block seed is the legal value for the block creator, that the receipts included are legal according to the receipt filtering rules, and others. Only after the block passes this set of validations will it be "chained" to the last block in each node database, and the transactions included in the block will be executed in the order they are listed. The execution of the transactions will thus change each single node's database state.

The Block Seed

ZooBC Proof of Participation consensus mechanism uses many strategies that depend on pseudo-random numbers which can also be agreed upon deterministically by the network (such that each node computes the same results of its operations without explicitly sharing these results to other nodes).

Because a strong hash function produces a uniformly distributed random number from arbitrary input data, the hash of a block may be erroneously used as a source of entropy. However, as a block creator only needs to rearrange the transactions in his block, or exclude some, or change the timestamp etc, to produce a completely new block hash, he may make many attempts to generate a hash which would be more profitable, and thus manipulate the blockchain, potentially opening the blockchain to attacks. Therefore ZooBC includes a special property in each block called the **block seed.** This property cannot be influenced by the block creator, and cannot be predicted in advance by anyone who does not have the private keys of the other block creators. The block seed is computed by the following formula:

$$\boxed{\text{block\_seed}_H = \text{creator\_signatureH}(\ \text{hash}(\ \text{block\_seed}_H -1\ )\ )}$$

In other words, the block seed is a digital signature by the block creator on the hash of the previous block seed. By using the Ed25519 digital signature algorithm, we guarantee that the signature of a message by some particular private key has precisely one resulting signature, removing any freedom on the part of the block creator as to which block seed he is required by the protocol to include in his block.

With this method, the only way a node can influence the block seed for the next block is to skip his turn (more on this in the next section) to create a block. The Proof of Participation algorithm pseudo - randomly assigns each node in the node registry a turn to create the next block, and by skipping his assigned turn, the block creator will lose participation score (if he does so repeatedly he will be ejected from the node registry.)

A lower participation score immediately affects the likelihood of a node to receive coinbase rewards and dividends of transaction fees. Therefore, exercising this one degree of freedom in influencing the block seed comes at a great cost, incentivizing even maliciously inclined nodes to submit a block seed on time which may be unfavorable to them.

VIDEO

Why Did You Choose the Ed25519 Digital Signature Type?

FORUM

Join discussions about Block Seed

Block Creator Selection

Because with the "node registry" we know the complete set of potential block creators (how a node is added to the registry is described below in the section on "node registration"), after each block, ZooBC pseudo -randomly generates a priority-ordered list of the nodes responsible for creating the next block. When a node misses its turn to create a block in this list (either because it is offline, not well connected, or deliberately skips its turn) its participation score is reduced, and the network awaits a block from the next node in the randomly ordered list.

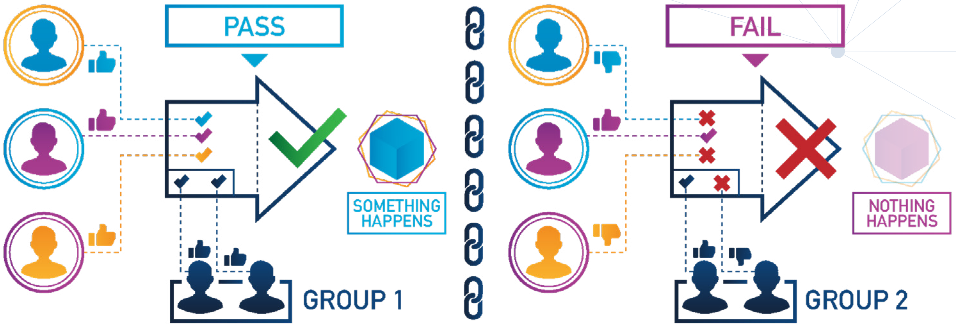
$$\boxed{order_N = hash(\ hash(BS_H) + PK_N\ )}$$

Cumulative Difficulty

In Proof of Participation consensus, the equivalent of "highest difficulty chain" in Proof of Work consensus is the chain whose blocks have been created most reliably by one of the highest priority nodes in the list of next potential block creators, which can be called the chain with the highest ***cumulative difficulty.***

To avoid attacks done by node administrators purposely leaving their nodes offline, to later rebuild a blockchain with a higher cumulative difficulty, the block creation can be done only by the first X (exact number TBD) nodes in the randomized priority list. If they are all unable to generate a block, any successive node can only produce an empty block. All the nodes before the one generating the empty block will lose participation score. If more than one block of the same height is proposed in the network, the nodes will choose the one generated by a node with a higher place in the randomized list. If two blocks are proposed from the same creator, the one with the earlier timestamp is selected. When a node is forced to evaluate two competing chains, it will always select the chain with the highest cumulative difficulty.

Multisig



When managing high-value digital assets, it is risky to have a single person holding the account key as the key of the account can get lost or compromised exposing the secured asset. Therefore we implement native multi-signature ("multisig") features, allowing users to create accounts that require X-of-Y signatures to perform transactions. For example, out of 10 people each holding a different key, a transaction can be performed by just 4 of those 10 signing it.

VIDEO

Explaining Multisig Mechanism

FORUM

Join discussions about Multisig

ZOOBC Q&A

Get your questions about Multisig answered

Multisig Addresses

One of the address types we support is a ***multisig address,*** which is the hash of a set of details regarding who is allowed to sign on behalf of the address. We refer to this set of raw details as a ***multisig info.***

This design leaves open the possibility that until the time of signing, the set of addresses which control a multisig address doesn't need to be revealed.

Multisig Info Object

Field	Description
MinSigs	The minimum number of signatures out of the provided address list which must be present to execute a transaction from this multisig address.
Nonce ("Access Code")	A free number field allowing unique multisig addresses to be created between the same set of addresses.
Addresses	An alphabetically-ordered list of account addresses that may legally sign for this multisig address. These addresses can, in turn, be multisig addresses, allowing for hierarchical multisig.

The Multisig Transaction Type

ZooBC implements all aspects of multisig behavior through a single transaction type. The behavior of this transaction type is somewhat complex because it implements both on-chain and off-chain multisig behavior, with the possibility to preserve the anonymity of the signers and conceal the transaction being signed until the moment that all needed pieces of information have been exposed to the blockchain.

The transaction body of a Multisig Transaction has 3 optional components: a Multisig Info object (described above) revealing which set of addresses may sign for the transaction; the transaction being signed on (either the unsigned full data of the transaction or its transaction hash), and a list of signatures on the transaction hash by other signers.

To manage the multisig process, the node keeps three consensus -managed tables: the Multisig Info table, the Pending Transaction table, and the Pending Signatures table. When a Multisig Info object is included in a Multisig Transaction, it will be saved in the Multisig Info table along with its hash (which is the corresponding Multisig Address.) When an unsigned transaction is included with a Multisig Transaction, the unsigned transaction and its hash are recorded in the Pending Transactions table. When some signatures on a transaction hash are included in a Multisig Transaction, a record for each is accumulated in the Pending Signatures table.

In this way a Multisig Transaction gets executed as soon as all the necessary parts (the transaction itself, the list of co-signers, and the needed signatures) are added to the blockchain by being included into validated blocks, independently of the order they have been received. After adding any transaction with multisig-related data to the relevant tables, each node will evaluate whether enough information is available to execute the specified Pending Transaction.

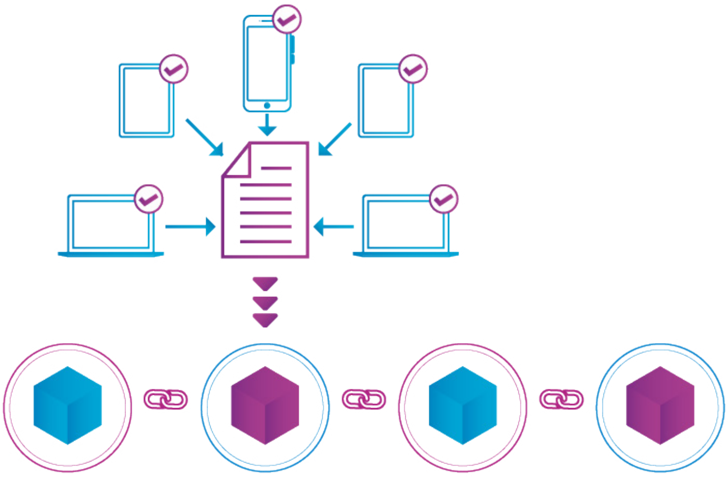
Specifically: if, for the specified Pending Transaction, the sender address is known to be a multisig address (because a Multisig Info object hashing to that address has already been revealed), there are enough signatures on its transaction hash in the Pending Signatures table (where each signature must be from an address specified in the Multisig Info), and the transaction is still valid at the time of application, then the Pending Transaction is executed as if it was a normal transaction on the blockchain occurring in place of the Multisig Transaction.

All three types of data have an expiry time, and so the Multisig Transaction will only execute in the case that these conditions are met in a timely fashion. Afterward, unused pieces of data will be pruned from the consensus-managed local tables of each node.

Multisig Use Cases

The purpose of the complexity of this transaction type is to give users the power through one tool to accomplish various flavors of multisig transaction behavior. For clarity, some of the use cases which may be satisfied by this design are described below.

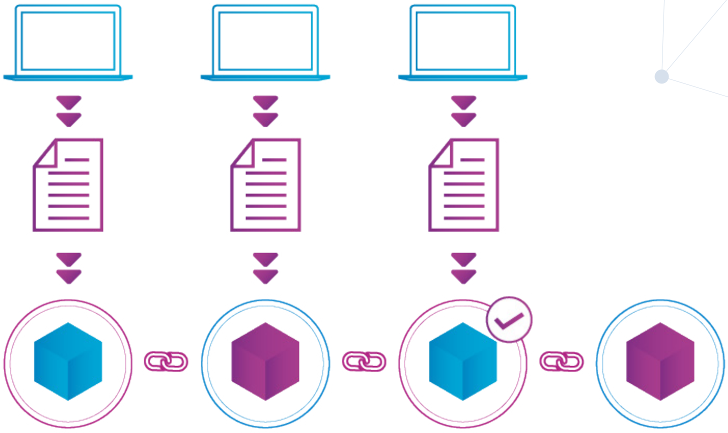
Off-Chain Multisig



In this case, the behavior of the Multisig Transaction approximates classical multisig behavior of Bitcoin and other blockchains, where the account controllers work together off-chain to prepare a single valid transaction that will be immediately evaluated and applied once broadcast to the network.

In the simplest case, all optional parameters of a Multisig Transaction may be supplied in one transaction, including the Multisig Info describing who may sign for the account, the transaction details to be executed by the account, and all needed signatures from the other account controllers.

On-Chain Multisig



This may be useful in cases where some participants wish to prove their existing signatures on-chain before others feel confident about providing their own signatures, or in cases when another required signer cannot be contacted off-chain by the other co-signers but may be alerted by some third-party application that a multisig transaction awaits his signature.

If the Multisig Info and Pending Transaction are already revealed, other account controllers may submit their signatures for the transaction as separate Multisig Transactions. In the extreme case, each needed signer may submit a Multisig Transaction appending only his own signature to the Pending Transaction hash, which will simply be accumulated in the Pending Signatures table until the number of signatures needed to execute the Pending Transaction is reached.

Anonymizing Multisig Addresses

This behavior may be useful in cases where it is desirable to not reveal the controllers of an asset until they must act, especially if they are in conditions where they must individually submit their signatures on-chain. Alternately this may be used as a mechanism intended to discourage someone from posting a transaction unless they really mean it: a Pending Transaction may be irrevocably committed (signed and ready to execute), giving anyone who possesses the Multisig Info the power to single-handedly force the transaction to be executed by revealing it.

Due to the structure of the Multisig Transaction, it is ok to submit a Pending Transaction from a multisig address, and all needed signatures on that transaction, before the Multisig Info (such as which addresses may sign for the transaction) are revealed. To further create plausible deniability for the actual multisig participants, many other accounts may blindly submit signatures for the given transaction hash, even if they are not co-signers of the multisig account. In this case, the Pending Transaction will be executed as soon as a Multisig Info matching its sender address is submitted in a separate Multisig Transaction.

VIDEO

Multisig and Key Management

Concealing Pending Transactions

This behavior may be useful when it is not convenient for the controllers of an account to pass around a partially-signed transaction, yet they do not want to reveal what action they plan to take unless it collects the necessary support of other account controllers (for example, a board of directors voting to fire the CEO, who can herself access and see the blockchain.) Like above, this may also be used as a privacy protection mechanism, where a Pending Transaction will be executed the moment any person who possesses it chooses to submit it.

Even if the Multisig Info is already revealed, the unsigned Pending Transaction itself may be hidden from the blockchain until enough signers have already submitted their signatures on its hash (in order to produce the correct signature, they must be given the contents of this transaction off-chain.) In this case, the Pending Transaction will be executed as soon as its full data is submitted in a separate Multisig Transaction.

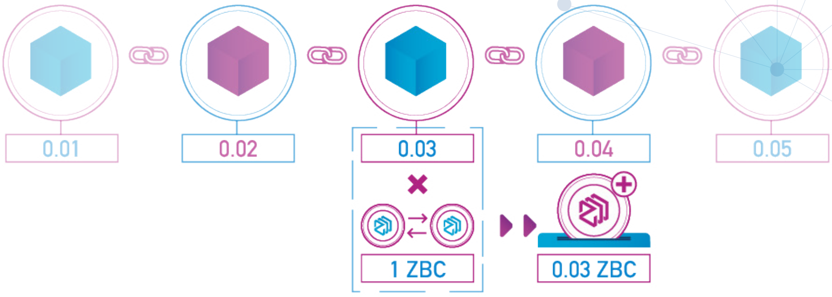
Hierarchical Multisig

This behavior may be useful when complex organizational structures are responsible for an asset. Say, for example, approval for a transaction is needed from 2 out of 3 departments, where each department, in turn, requires the approval of 3 out of 5 managers, and where perhaps a few of the managers represent oversight committees which themselves must cast a majority vote to approve.

Because a Pending Transaction may be any valid transaction on the blockchain, it may also itself be a Multisig Transaction. In this way, the creator of a multisig address may specify another multisig address as one of the controlling accounts. There is no limitation on how many multisig layers may be created this way.

In this case, a Pending Transaction may be created from the top-level multisig address. Before this transaction may be executed, a signature must be added by the controlling multisig address. The Multisig Transaction to add this signature becomes, itself, a Pending Transaction which must satisfy the conditions specified by its own Multisig Info before it will be executed.

Fee Scaling (Governance)



One difficulty with transaction fees on any public and permissionless blockchain network arises from extreme volatility in the value of the underlying token. Ideally the cost of fees to process transactions will remain approximately constant with respect to the stable currencies which users exchange for the token. For example, if the average minimum transaction fee is 1 token, this will fail to reduce network spam or incentivize block creators when the token value is US$0.0001, but it will likely make all transactions prohibitively expensive if the token value reaches US$100.

There are many strategies to approach this issue. On one extreme, the fee could be set by a trusted third party which signs such new information against a public key agreed upon by the entire blockchain; but such centralization is anathema to the ethos of decentralized systems, as it can be manipulated by a single actor who is beyond accountability. On the other extreme, nodes could attempt to draw independently from public sources, such as exchanges, the value of the token in a stable currency, in order to change the transaction fee amount to match a fixed value in stable currency; yet, because such data is not tracked by the blockchain’s own consensus, this could lead to forks when such external information is not consistent when queried by different nodes at different times.

Therefore we conclude that safely reaching consensus on data from outside of the blockchain, such as the token's value against other currencies, cannot be accomplished automatically. We implement a system by which operators of registered nodes on the network may take a regular vote on the appropriate multiplier, which we call the fee scale, for minimum transaction fees. This allows the transaction fee amount to keep the same value against a stable currency, even if the blockchain's main token value fluctuates.

It is potentially dangerous to put such a critical network parameter in the hands of node operators. A more complete discussion of the risk is presented in future work. However we prefer this risk over the inherent risk of imposing a static minimum fee for the reasons described above.

We prefer this risk because we believe it can be mitigated by balancing two competing incentives of node operators which should constrain each other: in the small scale, a node operator wishes to maximize the transaction fees he collects in each block by pushing the network fee scale higher; but on the large scale, network fees that are high will reduce user's willingness to pay for any transactions on the blockchain, and this lack of usability may be reflected in the token price, which determines the real value a node operator stands to gain, incentivizing him to push the network fee scale lower.

The vote-based adjustment of the fee scale is accomplished via three mechanisms: Node operators committing to their votes, node operators revealing the votes they have previously committed, and the averaging calculation used to compute the new network fee scale.

ZOOBC Q&A

Get your questions about Fee Scaling answered

Committing to Fee Votes

First the node operator needs to cast their vote on how much, if any, the fee scale should be adjusted. The fee is adjusted every month. Each of these one-month periods is further divided into two phases: a ***commitment phase,*** when the votes are cast, and a reveal phase, when the votes are revealed and counted. During the commitment phase, owners of nodes in the registry may submit a ***commit fee vote*** transaction.

First, the node operator's wallet will create a ***fee vote*** object. This object contains a recent block hash, the corresponding block height, the user's vote on what the new network fee scale should be, and the account's digital signature on the above information. Because we intend for node operators to vote in a way that stabilizes transaction fees against major fiat currencies, the wallet will collect this information from exchanges, calculate an appropriate value by which to multiply the fee and strongly recommend the value of the vote to the user. However, there is no way for the blockchain to validate this value, therefore it is only a recommendation, and the user may enter any value they wish.

The extra pieces of information in the fee vote object (block hash, account's signature) are included to make the hash of the fee vote object resilient to attacks where an attacker may simply guess what fee scale the user is voting for, hash their guess, and confirm their guess by comparing it to the vote commitment hash.

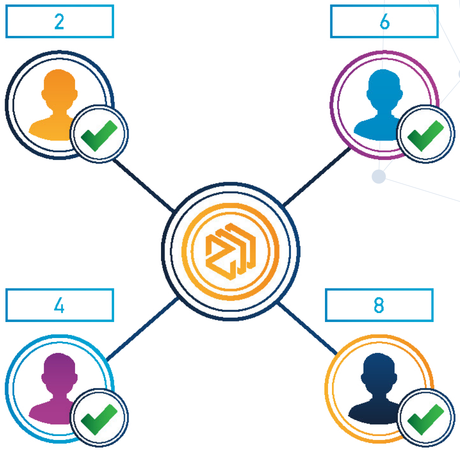
Because recent block hashes will be different for each voting period, and a digital signature is performed on these properties, anyone not possessing the account's private key is unable to reproduce the object and is therefore not able to reproduce the resulting hash of the object and determine by a guess-and-check method the vote being committed by a user.

Once the fee vote object is prepared, its hash is computed and included in the commit fee vote transaction, which is then signed and submitted to the network by the user. When this transaction is included in a block and applied, a record will be kept in the node's database indicating the commitment hash. Such transactions may only successfully be included in blocks during the commitment phase of the voting period.

Revealing Fee Votes

After the commitment phase of the voting period has concluded, the reveal phase begins. This is when the votes are recorded on the blockchain and accumulated. During this phase it is ok for the owners of registered nodes to submit a ***reveal fee vote*** transaction.

The reveal fee vote transaction includes the full contents of the fee vote object which the user previously committed to. It is then signed and submitted to the network. In order to be accepted by nodes as a valid transaction, the hash of the submitted fee vote object must match the commitment hash already submitted in the commitment phase by the user.



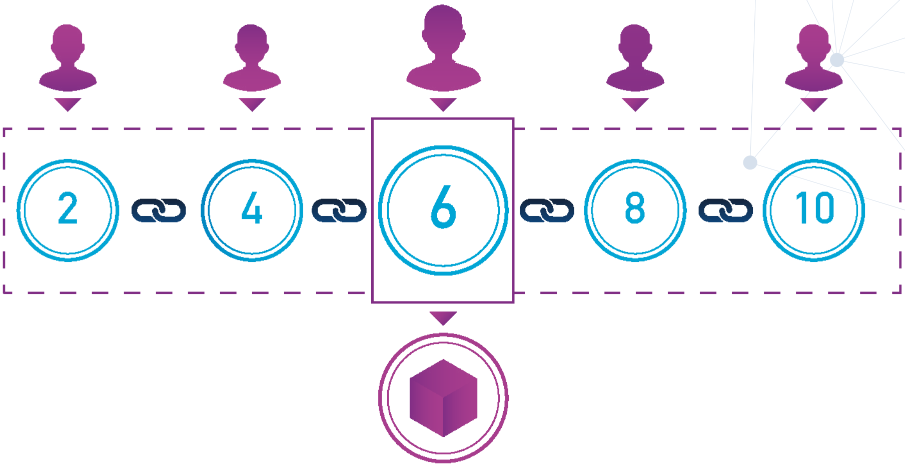
Additionally it must be verified that the "recent block hash" parameter is within the time frame of this voting period and that the digital signature on the parameters is a valid signature for the node operator's account.

When the transaction is successful, the user's vote is recorded in the blockchain. Values for new votes are accumulated during the reveal phase.

Adjusting the Network Fee Scale

The end of the reveal phase marks the end of the entire voting period, which immediately begins the next voting period by initiating another commitment phase. On the first block of the new voting period, a calculation is performed of all the successfully revealed votes from the previous voting period to determine the new value of the network fee scale parameter.

The collected votes for the new value are ordered from least to greatest, and the number in the middle of the ordered list is selected as the new value (the median value.) We use this approach, rather than averaging the votes, to avoid giving a small number of outliers the power to significantly bias the outcome. In the case that most votes are clustered together, even if a minority voted much higher or lower, we select the middle of the cluster as the most accurate representation of the majority opinion.



After this value is selected, it is further constrained such that it may only increase to a maximum of twice the previous value, or decrease to half the previous value. This reduces swings in the fee and creates a degree of predictability and stability for users.

This new network fee scale value is applied going into the new voting period. During this time, users can expect the parameter value to be constant, and wallets can query the nodes for the current value of the parameter in order to automatically recommend a permitted minimum transaction fee.

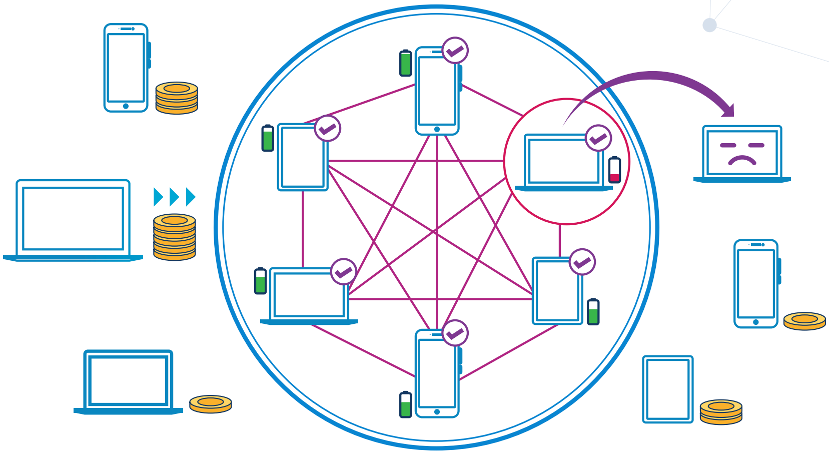
Depending on experiments, we may also allow a grace period for transactions created during the previous fee scale period to still be valid. We wish to avoid a condition where congested transactions in the mempool, which included a fee that was legal at the time they were submitted, are suddenly excluded as the result of a change in the minimum fee parameter.

Note on General Governance

While the description of voting mechanics in the above sections are specific to managing the "network fee scale" parameter, they could easily be applied to any other network parameter deemed safe to be placed in the hands of the majority owners of the node registry. These could include the maximum number of transactions per block, the average time between blocks, the rate at which new nodes are admitted into the node registry, or others.

Depending on the success of this limited form of governance mechanics in the first version of the ZooBC blockchain, we may apply this strategy to other constant parameters in future versions. This is consistent with our gradual and conservative approach to extending ZooBC through an iterative process of analysis, design and deployment. A list of constants that ZooBC needs to operate is at the end of this paper.

Node Registration



Although anyone can run a node and connect to the network, we restrict the creation of blocks and the distribution of coinbase rewards to a subset of registered nodes. The process of how new nodes can become registered, and how nodes are ejected from the registry, is managed entirely by the network protocol. This means that no single user or account has the authority to promote a node or change its status; nodes only change status by the automatic application of the protocol rules. This design has three key motivations:

- To prevent theft of private keys;
- To prevent an attacker from taking over a majority of nodes;
- To allow for the removal of unproductive nodes or those using a novel, unexpected, method to abuse the network

Following is a more detailed explanation of the purpose and mechanism for each aspect of the above. First, we separate the private key of the node owner's ZooBC account from the process of block creation. In blockchains where blocks are secured via digital signatures rather than a direct Proof of Work, the private key of the account of the user managing the node (the account that receives the coinbase rewards) must be present directly on the node, which is an online computer used to create blocks.

This creates a security hazard where the account's private key may be hacked from, or intercepted by, an attacker, if the computer hosting the node is breached or a Man In The Middle (MITM) attack is executed. For hosted nodes running in Virtual Private Servers (VPS), it could be as simple as the operator of the data center accessing the VPS file system and reading the private keys, giving them access to the account with funds.

Second, we regulate the rate at which nodes can join the family of block creators, in the node registry. To prevent attacks on the Proof of Participation algorithm where an attacker can register many nodes' public keys at the same time (thereby flooding the node registry and controlling a large majority of nodes capable of creating new blocks), the protocol rules only allow a limited number of new and active nodes to join the node registry during each set period. Restricting the registration of new nodes en masse preempts this attack.

Third, we can remove nodes that are offline (or simply don't participate in the network) from the node registry. The Proof of Participation algorithm described below has a scoring system that punishes nodes that don't participate. Any nodes which fall to zero participation score are automatically removed from the registry; if they later become active they are automatically added to the queue to re-join the node registry.

Beyond these motivations, it is also used as a weighting coefficient on a node's likeliness to collect coinbase rewards. This incentivizes nodes to remain online and participating. How this score is calculated is described in detail in the later section on Proof of Participation.

By maintaining a federation of nodes which continuously prove their active participation on the network, we create a foundation for the application platform we will deliver with the next version of ZooBC. Using the blockchain to reach consensus on the state of the registry will allow DApps to leverage faster federated consensus algorithms between registered nodes in future versions.

Although the node registry is not leveraged in its full capacity for this first version of the ZooBC technology, we intend this first deployment to gather information about its safety, the ease and intuitiveness of its management by node operators, and how it can be abused, in order to address any major issues before continuing to develop new strategies on top of it.

VIDEO

Fixing Critical Bugs of Node Registration

How Node Registration and Participation in ZooBC Work

How Can Users Register Their Nodes to the Wallet

What Do You Need to Run the ZooBC Node?

What Tips Could You Give to People Who Intend to Run the ZooBC Nodes?

What to Keep in Mind When Running the ZooBC Node

What Can a Node Runner Do to Make Sure He Always Stays in the Node Registry and Gains Rewards?

Why Do Nodes Need to Register Themselves on ZooBC Blockchain Network Before Finding Blocks?

FORUM

Join discussions about Network

Join discussions about Node Registration

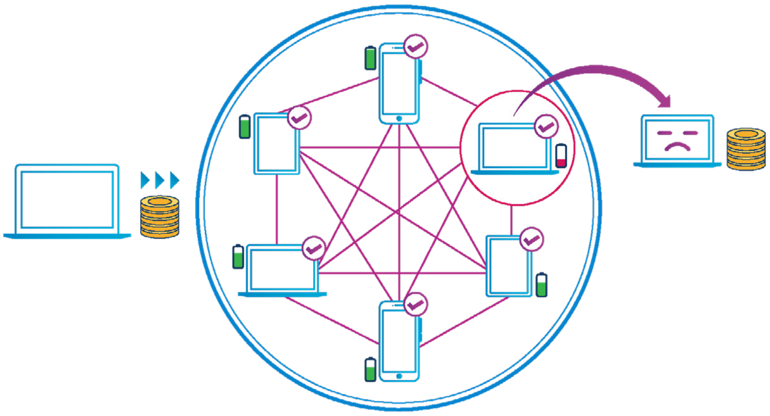
ZOOBC Q&A

Get your questions about Registration answered

Get your questions about Nodes answered

The Node Registry Life Cycle

Once a user has installed a node and allowed it to synchronize with the network, she may apply for a spot in the node registry by submitting a register node transaction. This transaction includes some details about the node, as well as a proof that the owner controls the node's private key.



Upon submission of this valid registration request, the node enters into the registration queue. The creation of newly available seats in the registry, and the replacement of nodes in the registry by those in the queue, are strictly regulated by the protocol rules. When seats become available, new nodes are admitted into the registry from the registration queue.

When a node is first in line in the registration queue, the protocol first requires it to prove its participation in the network for a brief period before admitting it to join the registry and begin creating blocks. If a node remains in the queue for more than 30 days without participating, its locked funds are returned to the node's owner account, and the node is removed from the queue. This means that if the node owner wants to add its node to the queue again, she needs to submit a register node transaction and pay the corresponding fee.

Once a node has been admitted into the registry, it becomes subject to the Proof of Participation algorithm, which will gradually modify the node's participation score upwards or downwards based on its behavior on the network. The registry seat also entitles the owner to participation rewards (coinbase), with the amount of rewards received being proportional to the node's participation score.

During the node's tenure in the registry, the node's owner may update her node's details as she sees fit by submitting an ***update node transaction.*** For example, if the node owner needs to change the node's key pair, or if the owner decides to increase her locked balance to increase her chances of remaining in the registry.

Additionally, if the node's owner loses her account's private key, so long as she still controls the node's private key, she may submit a ***claim node transaction*** from her new account to recover her locked funds, although this will remove the node from the registry.

This has the deliberate consequence that if an attacker can access your node's private key, he can effectively claim any funds you had locked to register the node. This forces node owners to take care of the security of their nodes, while making a node useless if hacked. The significance of this risk to the security of the proof of participation algorithm is discussed in more detail below.

A user may wish to remove their node from the registry deliberately to reclaim their locked funds, in which case they can submit a ***remove node transaction.*** This will credit any locked funds for the node back to the owner's account, and the node score is lost. There is no way to reclaim the locked funds of the node without simultaneously removing it from the node registry.

Finally, a node may be automatically ejected from the registry if its participation score drops to zero. In this case the funds locked with the node registration are still left with the node, that automatically joins the queue to re-enter the node registry, sparing a node owner in good faith to have to pay the fee to add the node to the queue again. As described above, as long as this node remains offline, it will repeatedly fail the trial period before it can be added to the registry again until it is removed from the queue.

Some aspects of this general process are discussed below in more detail.

The Node Registry

Each node on the network maintains a table we call the node registry. The registry serves first and foremost as a mapping between user account addresses and the nodes they operate, empowering the nodes to sign blocks on the user's behalf without exposing users private keys, and allowing the user's account to be directly rewarded for the node's participation.

Each node's entry in the registry declares the following properties:

Field	Description
Public Key	The public key corresponds to the node's configured private key. When blocks or Proof of Participation messages are produced by a node, the signatures can be validated against this key.
Account Address	The account address of the node owner's account. This account may legally change or remove the node registration, and any participation rewards earned by the node are credited to this account.
Locked Balance	An amount of funds which the node's owner has put up as collateral to compete for a spot in the registry, and to incentivize her to maintain the security of her node's private key.
Locked Balance	An amount of funds which the node's owner has put up as collateral to compete for a spot in the registry, and to incentivize her to maintain the security of her node's private key.

The Node's Public Key

Each node keeps its own private key, which is used to sign blocks on its owner's behalf. The corresponding public key is published to the network through the process described below, allowing other nodes to validate that blocks and peer-to-peer messages which originated from this node are authentic.

In the simplest case the node's private key is kept on the node's hard drive in a configuration file, however for node operators who are more security conscious, Blockchain Zoo is working to allow a separate hardware device to sign on behalf of the node. This is to mitigate the following vulnerability:

If a node's private key is compromised, the attacker may be able to impersonate this node on the network to others. However, as described in the Claim Node section below, the attacker will also be able to claim the node operator's locked funds and kick the node from the registry.

In the event that a user learns his node's private key may have been compromised, they can rotate the node's key pair and update his entry in the node registry at any time, by signing an update node transaction with his account's private key.

Locked Balance

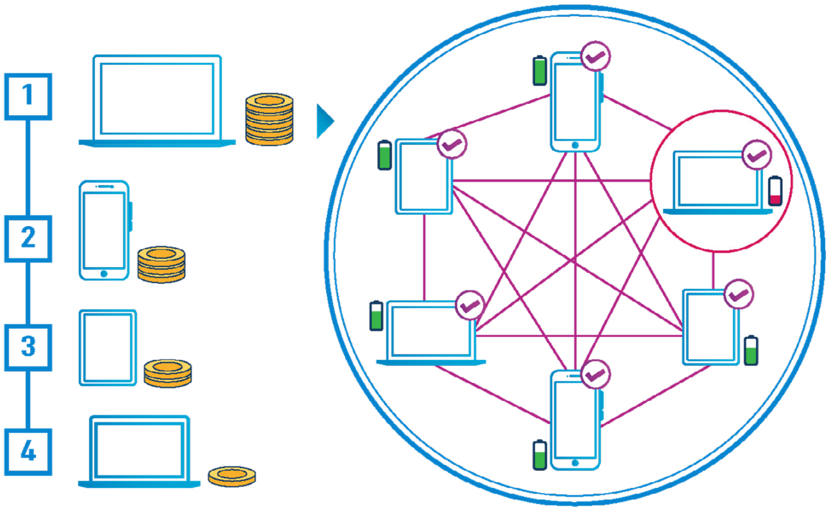
To maintain a node in the registry, the owner’s account must lock some amount of ZooBC tokens. When a node is added to the registration queue, this amount is deducted from the owner’s account and kept in trust by the network. Later, if the node is ejected from the registry queue, or if the owner removes it from the registry or the queue deliberately, the locked funds are returned entirely to the owner’s account.

Locked funds are used to prioritize the addition of nodes to the registry. New spots in the registry are only opened gradually, and the “size of a node’s locked balance is used to prioritize which node in the queue will take the next available spot. In this way, although ZooBC doesn’t use a variation of Proof of Stake consensus, a user’s funds on the network are still relevant to his ability to be admitted into the noderegistry in a timely fashion. In this first version of the ZooBC technology, we still require this fund locking as a mechanism to prevent Sybil attacks, as funds on the network are a scarce resource, and keeping them locked ensures that they cannot be used twice by the same actor.

However, we have confined the significance of a user’s balance to this particular corner of the protocol; in future versions of the technology we will apply an alternate Sybil prevention mechanism unrelated to the user’s account balance. We will substitute that without otherwise modifying the mechanics of the system. Doing so will fulfill a major long-term design goal of the ZooBC project: management, on a trustless network, of assets that have a value greater than the market cap of the native cryptocurrency of the blockchain, or even running a blockchain without native currency while at the same time guaranteeing its trustworthiness and security.

The Registration Queue

The registration queue is a list of nodes which have been identified by a node registration transaction, and whose owner has committed an amount of locked funds, but which have not yet been admitted into the node registry.



The queue is prioritized by the amount of funds locked by each registering account, such that the account which committed the most funds in its registration transaction will be the first in line to be added to the registry when new admissions are allowed.

New admissions are taken from the queue into the registry at a regular interval. The rate at which new nodes are accepted is a function of the existing size of the node registry, such that when the registry is small, new nodes will be added slowly, but as the registry grows new nodes will be admitted more quickly. This rate is fundamentally a security requirement of the Proof of Participation algorithm, as an attacker could trick the algorithm if he were suddenly able to take control of a major fraction of the registry. Admitting nodes gradually gives the algorithm enough time to remove cheating nodes before they can accumulate a majority. (See Node Registration, above)

Registering a Node

Before registering her node, the owner must collect a special message from it called a ***proof of ownership.*** This message simply contains the owner's account address, a recent block hash, the height of the block hash, signed off by the node's private key.

The proof of ownership message is then bundled inside a register node transaction, together with the node's public key, the amount of funds to lock from the sending account, the account's signature, and the fee for the transaction. To prevent malicious users from abusing the register node transaction, the fee to add a node to a queue is much higher than most transaction types.

When this transaction is executed by the network, the balance specified by the owner will be locked, and the node will be added to the registration queue. The locked funds are taken from the user's account and held in trust by the network, until such time as the node exits the queue or registry for any reason, at which time the funds are returned to the user's account in full with a 1-day delay.

The wallet application we provide helps to smooth the process of collecting the proof of ownership message from the user's node and assembling the transaction for the network, ensuring the user has an intuitive interface and needs no specialized technical knowledge of the system to perform this operation.

Claiming a Node

By design it is not particularly dangerous if a user loses control of a node's private key, since it is only used to create blocks, and to prove a node's identity in the Proof of Participation algorithm. As described above, this isolation is a key motivation for separating the node's private key from an account's private key.

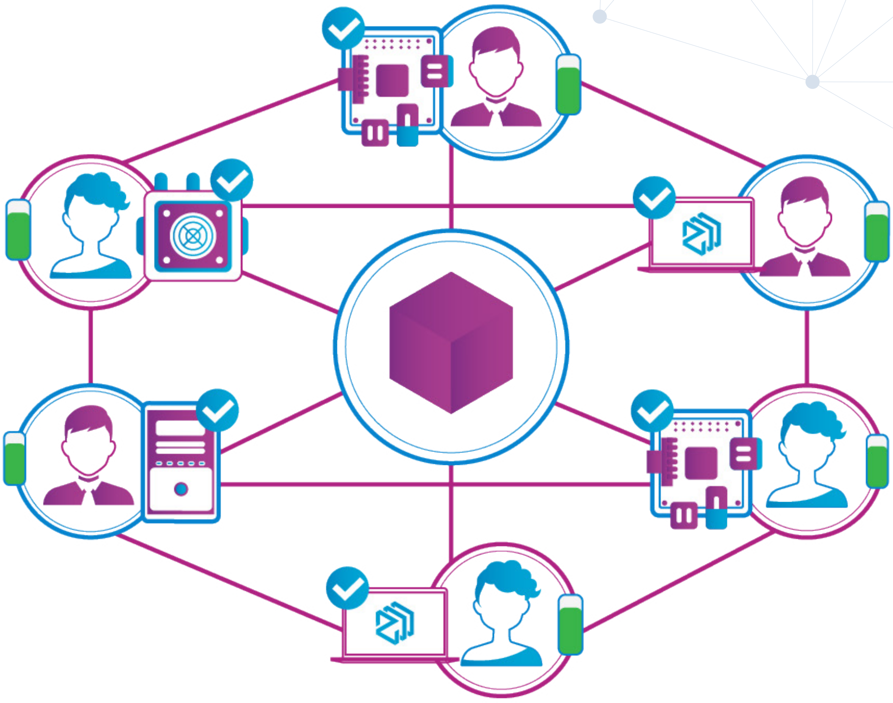
However, there is a type of "key sharing" attack on the Proof of Participation algorithm where all nodes may voluntarily share their private keys with each other. If all users agreed to share their node keys, they would undermine the central assumption the algorithm is premised on: namely that only a user's node may generate a valid digital signature, proving the node was online and participating with the network at the time.

Therefore, in the case that a user comes into possession of another node's private key, ZooBC allows this user to "claim" the node's locked funds, which will also eject that node from the registry. Such a "key sharing" attack then requires enormous trust in the other participants in the scheme not to steal funds from each other, while still allowing a user to limit the damage that can be done to them by only locking in their node an amount of funds they are willing to put at risk. This incentivizes node operators to take seriously the security of their nodes, as nodes are custodians of the private keys that control such locked funds.

Ejection from the Node Registry

If at any time the participation score of a node in the registry drops to zero, the network's consensus rules dictate that the node will automatically be returned from the registry into the queue. In this way a node must be consistently online and participating to remain in the registry and collect coinbase rewards. Ejection from the queue does not penalize the user's locked funds, which are returned in full to the node owner's account.

Proof of Participation



We propose a novel consensus algorithm that involves proving that registered nodes are reliably online and propagating data to each other, computing a participation score for each, and using this score as a weighting coefficient to the node's pseudo-random chance to be elected to receive coinbase rewards. If a registered node's participation score falls to zero it is automatically ejected from the node registry.

Our intentions with this strategy include promoting the availability of network nodes, more evenly distributing stewardship of the blockchain history among many parties, and more evenly rewarding the participation of all nodes composing the network. Additionally, we create an incentive scheme for the network of all registered nodes to organize itself into an optimal topology which minimizes the number of hops any piece of data must take to propagate through the entire network.

VIDEO

- What Is Proof of Participation in Simple Words?
- How Does the Proof of Participation Algorithm Work?
- Proof of Participation VS. Other Consensus Algorithms
- Why Does ZooBC Blockchain Use a Proof of Participation Algorithm?

FORUM

- Join discussions about Core General
- Join discussions about Proof of Participation
- Join discussions about Consensus

ZOOBC Q&A

- Get your questions about Participation answered
- Get your questions about Proof of Participation answered

Overview

“Participation” could be defined in many ways, but for the purposes of this algorithm we mean that a node is transmitting pieces of blockchain -related data (such as blocks and transactions) to many other nodes in a timely fashion. While tracking the entire set of node-to-node transmissions is infeasible, we use a random sampling strategy to pick some small subset of transmissions from the network to evaluate who is participating.

Such proofs of transmission would be meaningless if they could be produced on demand; therefore, nodes must regularly publish “commitments” of their transmission activity to the network. When a node is called on to produce some small sample of its past transmission activity, it should also prove that the records it produces were included in commitments already published on the blockchain, which makes such records impossible to produce just-in-time.

The ZooBC network protocol specifies that each transmission should return a ***receipt:*** a special object, digitally signed by the receiver, which uniquely identifies the sender. The receipt is a claim that a particular piece of data was transmitted between those nodes at that time. A receipt object also includes a ***commitment:*** the Merkle Root of a set of other receipts which had previously been collected by the receiver.

When a node creates a block, it can include some receipts of its past transmission activity. These receipts can be validated objectively by all nodes using several criteria. The number and quality of the included receipts are used to calculate whether the node's participation score should rise or fall.

Because we wish to measure only recent activity on the network, we impose a ***height filter*** on receipts which may be included in a block, such that receipts expire some number of blocks after they were created. The expiration time is a function of the number of registered nodes, because a greater number of block creators means a greater average time between blocks created by any given node, which consequently expands the average time we expect between a receipt's publication and the time it was previously committed.

Because we wish to prove the node has received and propagated the full set of data items on the blockchain, we impose a ***data filter*** on the receipts which may be included in a block. Based on the number of data items recently included in the blockchain, we pseudo-randomly restrict the set of data hashes which may be legally included, in a way that is not predictable before the block creator's turn. This ensures that a node must keep receipts from all data items transmitted in order to reliably produce the randomly-selected subset.

A receipt is only worth a higher participation score if the publishing node can prove that the receipt was a member of a commitment (Receipt Merkle Root) already included in a previously published receipt, which proves the new receipt that matches these filters existed before the filter criteria were known. When such a proof accompanies a receipt in a block, it is defined as a ***linked*** receipt.

If a node does not have any receipts to link, he can still include un-linked receipts (not included in a Merkle Root published by another node in a previous receipt) to earn a much lower score. While these receipts prove little about the publishing node, they do include commitments from their creators, which can later be used to prove pre-existence when those creators publish and link their own receipts.

In order to compute the change in participation score, we assign each block a value by counting a small number of points for each un-linked receipt and a larger number of points for each linked receipt. There is a fixed maximum number of receipts which can be included in a block, so we can clearly state that the maximum block value is given for a block filled with linked receipts, and the minimum block value (0) is given for a block with no receipts.

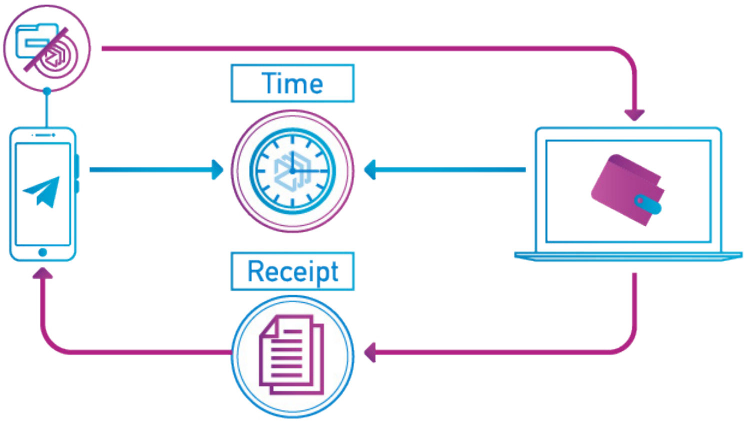
If the total value for the block is more than half of the maximum block value, the creating node's participation score will increase for this block; otherwise it will decrease. In the case that a node misses its turn to create a block entirely, it will forfeit twice the amount of a participation score as if it had produced a block with zero receipts.

The Receipt Object

Field	Size	Description
Sender Public Key	32 bytes	The public key of the sending node
Receiver Public Key	32 bytes	The public key of the receiving node
Data Type	4 bytes	A code indicating the type of datum that was sent. (Block, Transaction, File Chunk, etc.)
Data Hash	32 bytes	The hash of the Datum that was sent
Ref Block Height	4 bytes	The height of a recent reference block
Ref Block Hash	32 bytes	The hash of the recent reference block at the height specified
Receipt Merkle Root	32 bytes	A Merkle root of receipt objects previously received by the sending node
Receiver Signature	32 bytes	The digital signature of the receiving node (receipt producer) on all of the above data
Total	**200 bytes**	

Producing Receipts

When a node transmits some piece of data to another node (such as a transaction or block), the receiving node (after validating the data) should produce a receipt object and return it to the sender. This receipt should be created and returned regardless of whether the receiving node had already received and rebroadcast the same data from another node, so long as the data itself is valid.

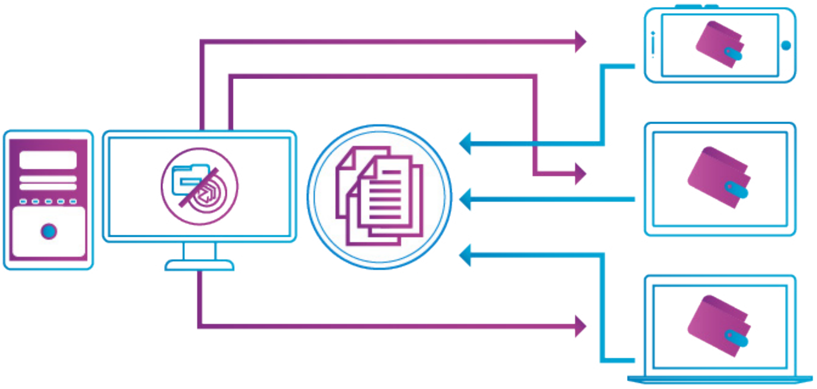


The receiving node will populate the "Sender Public Key" with the public key of the node which sent it the data, and the "Receiver Public Key" with its own node public key. "Data Type" is filled with a type code stating the data is a block, transaction or potentially other kinds of transmission, and the "Data Hash" field is filled with the hash of the data item which was transmitted. "Ref Block Height" is filled with the node's current block height, and "Ref Block Hash" with the hash of the block at that block height.

To fill the "Receipt Merkle Root", the receiving node will look up a recent Merkle Root in its "batch table" (described below.) While the content of this Merkle Root field cannot be validated by other nodes, it is in the receiver's interest to include a proper commitment which he can later use to prove the prior existence of other receipts he has collected.

Finally the node signs all the above data with its private key, guaranteeing the receipt could not be produced without his involvement. The receipt object is then sent back to the node which transmitted the data. Repeated failure to return a valid receipt object may result in the sending node blacklisting the receiving node.

Collecting Receipts



As a node broadcasts pieces of data to other nodes, it will collect and save the receipts from each receiving node that are returned. These receipts are organized into ***batches,*** where a Merkle Root is calculated for each batch which can be included in future receipts produced by the node.

The number of receipts which we allow to be proven by a single Merkle Root is limited, therefore it is not in the node's interest to save any receipts which it already knows will not be usable later. The node can safely discard any receipts which it already knows will not be permitted to include in a future block, in particular receipts which do not match his peer filter.

The maximum batch size is governed by a constant defined in the protocol, ***rmr max depth,*** which is the maximum allowed depth of a Receipt Merkle Tree. Functionally, this translates into the maximum number of intermediate hashes a node is allowed to publish along with a receipt to prove its membership in a previously published Merkle Root.

Receipts only need to be collected in memory by a node (i.e. not written to the hard drive) until the node is ready to finalize the batch and then write them to the database. When it is time to finalize the batch, the node will first compute the Merkle Root of all the receipts in the batch and save a new record in the batch table connecting the root to the block height at which it was created.

The node then adds all of the receipts included in the batch to the receipt table. Each record in the receipt table will specify the Batch ID (Merkle Root) that the receipt belongs to, and also its Sequence Number within the batch.

In this way, as the node collects receipts, it keeps a personal record of all the information it will later need to find out if one of its commitments has been published by another node and to construct the necessary proof (or "link") that some of its receipts were previously committed.

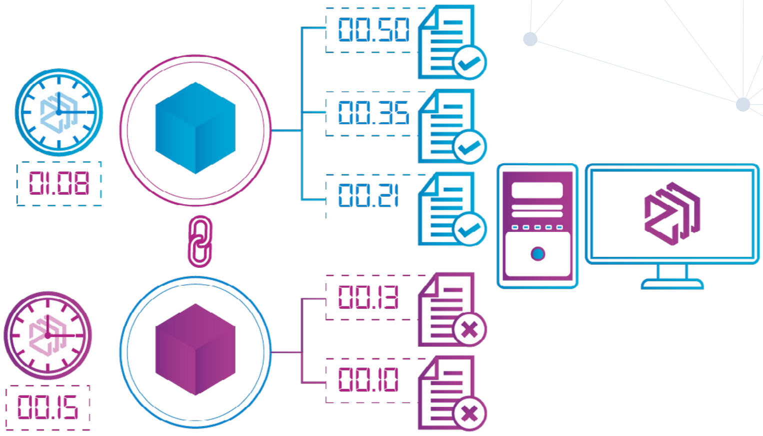
Batch Table Structure

Field	Type	Description
Batch Merkle Root	32-byte blob	The Merkle root of all receipts included in this batch.
Created Height	4-byte int	The block height when this batch was created.

Receipt Table Structure

Field	Type	Description
Batch ID	8-byte int	The first 8 bytes of the Batch Merkle Root
Seq Number	4-byte int	This receipt's position within its batch
Receipt Hash	32-byte blob	The pre-calculated hash of the receipt object
[Receipt Data]		The full data of the receipt object structured the same way as the receipt object specified above.

Pruning Old Receipts



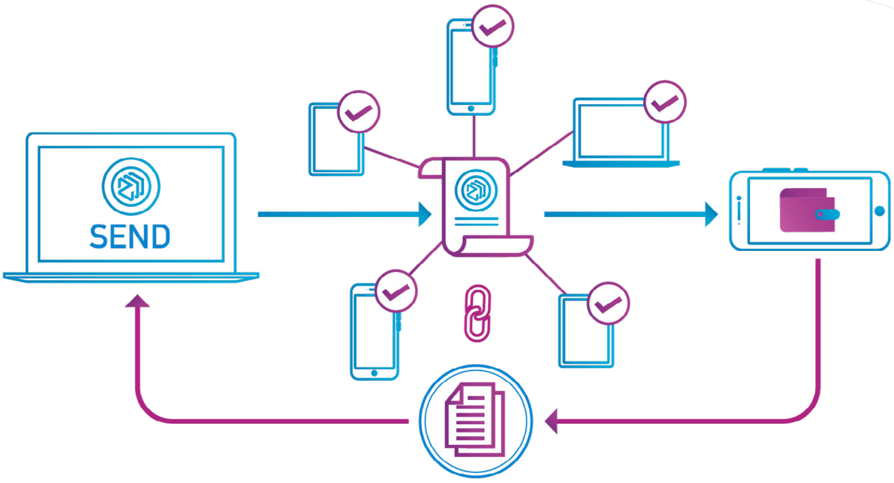
As all receipts have an expiration time (based on the block height they include), there is no benefit in keeping them forever. Additionally, each batch stores the block height from when it was created, making it easy to detect when all the receipts within a batch have expired. These expired receipts can be safely deleted.

The node will run a pruning process which occasionally checks if there are expired batches, and clean them from the database. Therefore, the entire receipt storage process should fit in a constant memory footprint, cleaning the old ones as new ones are accumulated.

Proving Linked Receipts

When a receipt is included in a block, it can either be un-linked (meaning its prior existence cannot be proven by any Merkle Root previously published) or linked (when prior existence can be proven.)

When a node wishes to link a receipt, it must include a set of intermediate hashes which, when hashed in sequence with the hash of the receipt, yield a Merkle Root hash which has been previously published in a block.



In order to do this, when constructing a block, the node will first compare the set of Receipt Merkle Roots it has stored to all recently published Receipt Merkle Roots, and begin searching through any receipts it possesses which are already included in one of these previous Merkle Roots. When it finds a receipt that matches the filter criteria for receipts in the new block it is creating, the node can use the precomputed set of intermediate hashes composing that Merkle Root to look up which intermediate hashes it must include with the new receipt to prove the linkage.

In practice, proving the link is quite straightforward: first, hash the receipt. Then, find the hash of the string made of that receipt hash concatenated with the first provided intermediate hash. Then, hash the obtained result concatenated with the next provided intermediate hash. Repeat this step for all provided intermediate hashes. If the result at the end precisely matches a recently published Merkle Root, this proves the receipt object must have already existed at the time that Merkle Root was created, and the link is valid. If it matches none, then the block creator has tried to forge a link where none exists, and the entire block becomes invalid.

The Height Filter

As described above, each receipt to be included in a block must be valid according to different filters, which are parameterized when the block is being created and can be objectively validated by other nodes when they receive the block. The simplest of these is the ***height filter.***

Each published receipt contains a block height and the corresponding hash of the block at that height. It is possible to craft a receipt that specifies an earlier block height than when it was produced, but it is not possible to produce a valid receipt for a future block height, as this would require foreknowledge of the block hash at that future height. Therefore we can say that a receipt was created no later than the block height at which it was created and signed.

Each time a node is added or removed from the registry, the network recomputes the receipt expiration time, which is the maximum difference allowed between the block height at which a receipt was created and the block height at which it is published. When a receipt is selected to be published, the node will compare its age to the receipt expiration time to determine its validity, and other nodes will confirm this when they validate the receipts published in the new block.

In order to maintain or increase its participation score, a node must reliably publish linked receipts. To give a node a fair chance to survive in the registry, the receipt expiration time must be greater than the average expected time between a previous block publishing one of its receipts, and the node being allowed to publish a receipt that links to it.

For this reason, the expiration time is computed as a function of the registry size, as the number of nodes in the lottery to create blocks can be used to determine the average chance that a certain number of receipts from a node have been previously published within a given timeframe. We will continue to experiment with this function through our alpha and beta network testing to reach the best algorithm to keep this timeframe secure.

The Peer Filter

The most complex filter imposed on receipts is the ***Peer Filter.*** The first objective of this filter is to require that each node in the registry connects to a diversity of other nodes and that the selection of these preferred peers is random and beyond its control. This minimizes an attacker's ability to selectively favor his own other nodes with the Proof of Participation algorithm, as on average he would need to control almost all of the registry and withhold his participation from all of the remaining nodes in order to eventually create a more valid chain.

We divide time into ***network topology periods*** of 60 blocks. Every 60 blocks, the network will pseudorandomly compute an ordering of all nodes in the registry, and use this ordering to assign a set of preferred peers to each node. Each node can easily compute its own preferred peers and connect to them in favor of others. More importantly, when the node publishes receipts later, the receipts it is allowed to publish must come from one of its assigned peers at the time the receipt was created.

The second objective of the filter, closely connected with the first, is to ensure that each node in the registry has a fair chance to have their previous receipts published. This helps create some statistical certainty and uniformity in how many blocks we can expect on average between any given node's receipts being published, which helps guarantee that even honest nodes do not make their receipt selections in a way that accidentally excludes some other honest node.

Beyond the above properties, this preferred peer assignment strategy also gives us the opportunity to optimally organize the peer-to-peer network topology of nodes in the registry. By optimally, we mean that assuming all nodes are online, for a network size N, and a number of preferred peers P, we can guarantee that a new transmission is gossiped to the entire network in a maximum of H hops, computed:

$$\boxed{H = \mathrm{ceil}\,(\,\log_P\,(\,N\,)\,)}$$

For example, if each node has 20 assigned peers to broadcast to, even in a registry of 50,000 nodes, the broadcast will reach all nodes after only 4 hops.

While decentralization prevents us from forcing any given peer to follow this preferred peer assignment, it creates a strong crypto-economic incentive for nodes to comply with this connection strategy: if they do not collect receipts from their assigned peers in a given time period, they will be unable to publish receipts later which pass the peer filter, and subsequently their participation score will fall, reducing the amount of coinbase rewards they will collect and eventually ejecting them from the node registry altogether.

The Data Filter

The two filters above are known in advance to the node collecting receipts, and a node could pass them by only transmitting a few pieces of data to each of its peers during each network topology period. Because we wish to incentivize nodes to transmit all relevant data, we impose a ***data filter*** on the receipts that can be published in a block, such that only receipts for a small random subset of all data recently transmitted may be included in a given block.

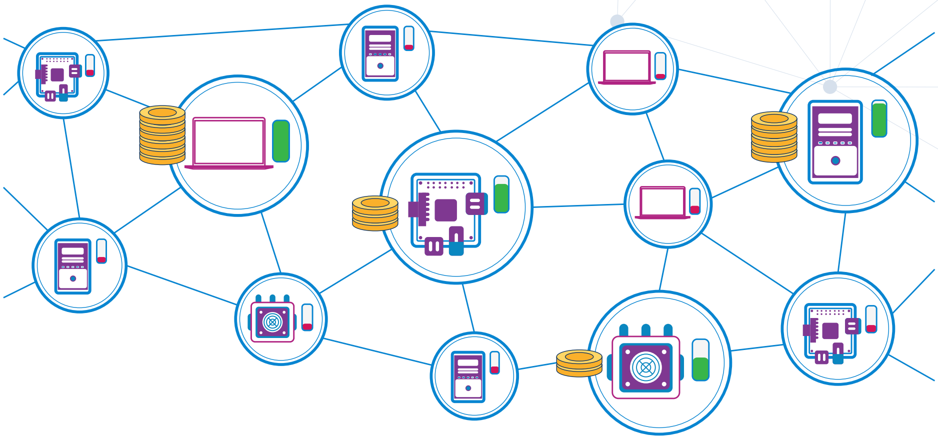
In practice, the node keeps count of the number of blocks and transactions that have been published since the receipt expiration time. (This counts only transactions included in blocks, so the number for a given block height is strictly in consensus for all nodes.) This number is used to calculate a filter width, a number between 0 and 1, which represents the likelihood that the hash of any given block or transaction is good to be published in a new block.

A receipt passes the data filter if, for the BlockSeed and FilterWidth at block height H, and for the DataHash of a receipt R, the following condition is true:

$$\boxed{(\ \mathrm{hash}\ (\mathrm{BlockSeed}_H + \mathrm{DataHash}_R\ )\ [\ 8\ ]/(\ 2\verb|^|64\ )\ ) < \mathrm{FilterWidth}_H}$$

Therefore, in order to have a reasonable chance of being able to provide enough receipts which pass the data filter when called on to produce a block, a node must collect receipts for all data transmissions which occur during this period.

Coinbase Distribution



Conventionally, only the node which produces a block on the network is rewarded with newly minted tokens. If the network becomes very large, the chance that any given node will produce a block (especially one with a lower participation score) within a specific period, and therefore receive a reward, grows small

One of our major objectives with the Node Registration and Proof of Participation algorithms is to more fairly reward the full set of network participants, and to do so in a more timely manner. In keeping with this goal we implement a pseudorandom lottery to reward many accounts per block, with participants' chance to win being weighted by their participation score.

FORUM

Request Testnet Tokens

Join discussions about Coinbase

ZOOBC Q&A

Get your questions about Coinbase answered

Get your questions about Coins answered

Here we detail how we compute the amount of new tokens minted per block, and how they are distributed to network participants.

Coinbase Schedule

Styled after Bitcoin, many blockchains offer a fixed reward per block for some period of blocks, after which the reward amount is cut in half for the next period, and so on. This geometric reduction ensures that the earlier participants are rewarded more than the later ones, and also that the number of tokens produced will approach, but not exceed, a target total supply.

We feel it is cleaner to define a smooth curve across all blocks rather than explicit halving events, such that the number of new tokens produced by a block is a simple function of its block height. For this curve, we take a window of a common sigmoid function:

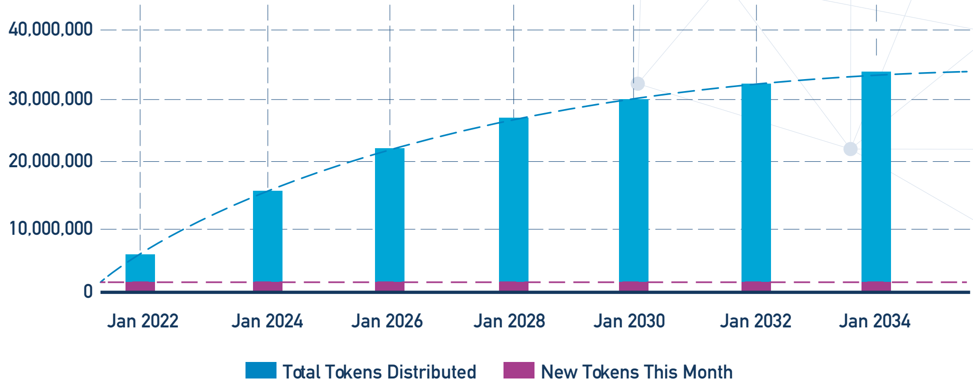
$$\boxed{1 / (1 + e\verb|^|(-x))}$$

Where **x** ranges from approximately **2-6**, while we still evaluate the parameters. This window is stretched over a period of **180 months (15 years)** to give the coinbase curve.

We prefer a function of this form over a pure logarithmic curve (which other networks such a Bitcoin approximate with periodic halvings) because as a pure curve, this would yield an inordinate number of coins released in the first few years, which we would prefer to leave for later participants.

Based on this rate of distribution, ZooBC expects to reach a target supply of 33’333’333 tokens over 15 years (the exact values will be evaluated and announced before the Beta version goes online.)

Total Distributed VS Per Month



Example token distribution based on the Logistic function, using window x={3...6}

Use the interactive Google Sheet and suggest to us the best curve by copying the document and editing the fields with a yellow background - ***here.***

Recipient Selection

For each block, a list of coinbase recipients is computed deterministically from the current state of the Node Registry and the new block's seed. We define an ***ordering function*** to compute a pseudorandom number for each node, weighted by its participation score, then select up to a maximum of X nodes with the lowest computed order numbers.

Where hash is the first 8 bytes of a SHA3-512 hash, PK_N is the public key of node N, PS_N is the pop score of node N, and BS_H is the first 8 bytes of the block seed at height H, we give the ordering function as:

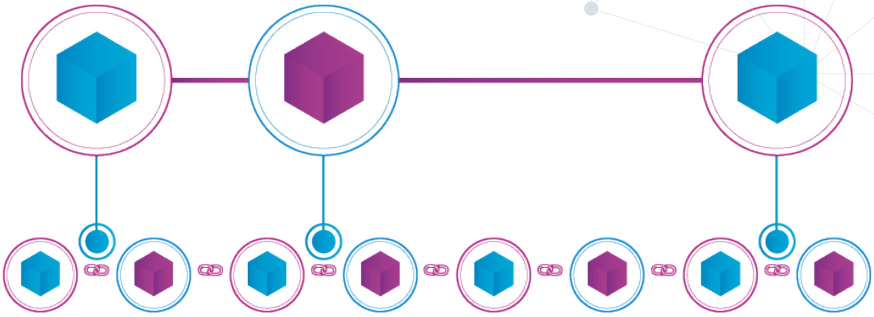
$$\boxed{order_N = hash(\ hash(BS_H) + PK_N\ ) * (1 / PS_N)}$$

Once we have computed this list of winners, the total block reward available according to the provided Coinbase Schedule function is divided evenly between the winners, with the newly minted tokens being credited to the winning node's associated account.

It is worth noting that the Proof of Participation score has a fixed maximum, which any good node should attain after some time. Therefore in a large network of good nodes, rewards should be frequent, fairly distributed, and not directly tied to recent block production, given that the node keeps participating.

Why does ZooBC blockchain use **SHA-3** hashing algorithm instead of **SHA-2**?

Spine Blocks



The ZooBC architecture creates once per day a special block called a ***spine block*** that is also chained to the previous spine block. Those blocks form a set of hops that allow a node to skip regular blocks following the spine blocks from the Genesis block to the current time. This provides a fast route to any moment in the history of the blockchain. Spine blocks are created once per day on average, and contain the major updates to the node registry (which nodes joined and which left) and other metadata, but contain no transactions, resulting in an extremely light set of blocks to download.

In this way, a node only needs to download a very light block for each day of the life of the blockchain, having a constant up-to-date list of who was in the node registry at each moment in the history of ZooBC, and so can evaluate if the next spine block has been created by a legitimate node. This allows, in the case of a fork, a new node to choose the best spine block between several options presented.

The first step for any new node is to download, starting at the genesis block, all the spine blocks (choosing, in case of a fork, the set of blocks with the highest cumulative difficulty) until it arrives at the latest available spine block. The node then uses the most recent blockchain snapshot hash found in a spine block (more on this below) to identify and download a blockchain snapshot from peers on the network. This allows a new node to come up to date with the live blockchain in a few minutes, even if ZooBC has run for decades and the blockchain has a collective weight of many gigabytes of data.

VIDEO

- Spine Block Development
- What Is a Spine Block (earlier called a Mega Block)?
- Relation of Spine Blocks and Snapshots
- How Does Spine Block Make the Blockchain Download Time Faster

FORUM

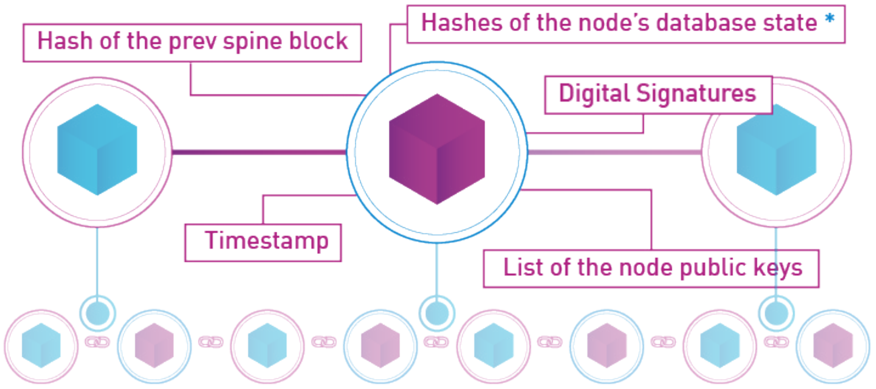
- Join discussions about Block Types
- Join discussions about Block Backups
- Join discussions about Rollback

ZOOBC Q&A

- Get your questions about Spine Blocks answered

Structure of a Spine Block

Unlike regular blocks on the blockchain, in which the block creator can select which transactions and receipts will be included, the contents of a spine block (all except for the set of digital signatures needed to create and validate the spine block) is purely determined by the state of the blockchain. Therefore any node in consensus will produce precisely the same spine block that any other node would produce at the same block height.


Hash of the prev spine block
Hashes of the node's database state *
Digital Signatures
Timestamp
List of the node public keys

* All the other days of the month no snapshot hash is included in the spine block.

A spine block first contains the hash of the previous spine block, and a timestamp, just like a normal blockchain's block. This is the essence of any blockchain and guarantees that the older blocks have not been tampered with.

Each spine block also contains a list of node public keys which have been added to, or removed from, the node registry since the last spine block. As the node applies spine blocks in sequence, the set of additions and removals tracks a "key pool" which roughly follows the set of node public keys in the node registry.

Each spine block must contain a collection of digital signatures on its contents. The set of keys which may legally sign, and how much value each of their signatures contributes to the cumulative difficulty of the spine block, are governed by the consensus mechanism described below.

Finally, approximately once per month, based on the rules for when database snapshots are taken (see below), it is ok for the spine block creator to include the hashes of the node's database state. All the other days of the month no snapshot hash is included in the spine block. Details on how this snapshot is created are presented in the section below on Snapshots.

Signature Accumulation

Every node in consensus will generate the same spine block at the same moment, so we need a mechanism to govern which node will broadcast the spine block first, and how it will accumulate signatures from other nodes in the key pool.

In the same way that potential next block creators are selected from the node registry, after each spine block, a priority list of next signers is calculated. To be valid, the new spine block must collect the signatures of a large number of these nodes. The higher on the priority list the signers, the greater the "cumulative difficulty" of the spine block. In this way only an attacker which controls more than 90% of the nodes in the registry can be lucky enough to create a set of spine blocks with a greater cumulative difficulty (and therefore more authoritative) than the honest set of spine blocks.

Nodes then undergo a process of gossiping signatures on the new spine block to each other, until enough are collected to consider the spine block confirmed, and assign it a cumulative difficulty score according to the priority position of the signers in the randomized list. At this time, the hash of the new spine block is finalized, and it is broadcast along with all collected signatures to the rest of the network.

Just as with regular blocks, if there are two competing versions of the spine block sets available, the node will always select the one which has the higher cumulative difficulty score.

Joining the Network

When a new node connects to the ZooBC network, it will first reach out to the well-known peers it is configured with. From these, it will continue through a period of network discovery by querying peers from elsewhere until it has a sizable collection.

The node will first query the hash and height of the last spine block from this set of peers. If there are multiple candidate sets of spine blocks, it will first select the set which reports the highest cumulative difficulty. The node will then download all the spine blocks from its connected peers, starting at the genesis block, confirming that the cumulative difficulty claimed by the last spine block is legitimate.

After arriving at the latest spine block, the node looks backward to find the latest registered snapshot hash. The node can then compute (as a function of the snapshot hash and the current state of the node registry) which nodes maintain this snapshot, and begin downloading it from them.

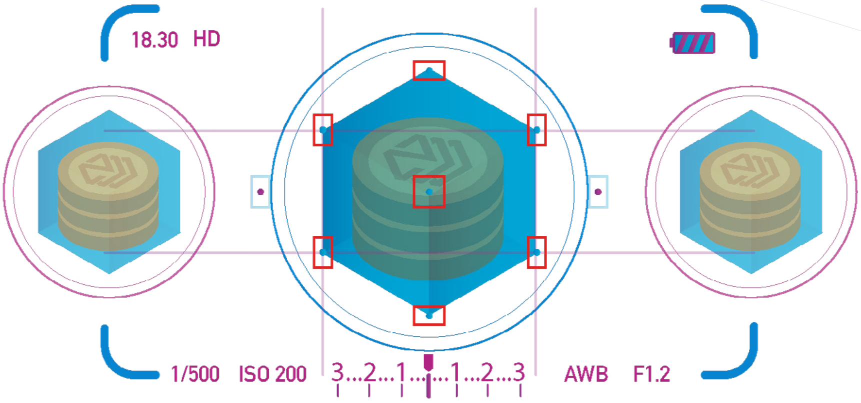
Once the snapshot is downloaded and hashed to confirm its legitimacy, the node will import the contents into its current database state. This brings the node up to date with the state of the blockchain at the block height when the snapshot was created. From there, the node simply downloads from the network the remaining blocks as usual to catch up to the current state of the network.

In other blockchains the blocks that a node needs to download to validate the current state of the blockchain are those with all the transactions from the genesis (block 0) to the latest block height. In ZooBC, a node can shortcut to the most recent archived state through the spine blocks, and only download the full blocks of the time since the last snapshot (which at maximum would be a month worth of blocks).

VIDEO

What Nodes Can Validate Spine Blocks?

Snapshots



One of our objectives is to reduce the blockchain download time to a constant (or nearly so.) The blockchain download time is a notorious problem, as traditionally each node must download and apply in sequence the entire history of previous blocks and transactions before it can begin evaluating the validity of new transactions, and the size of this historical record necessarily continues to grow for the lifetime of the chain.

To give an example: at the date of writing (18th of November 2019), the Bitcoin blockchain, almost 11 years old (the genesis block is dated 9 January 2009), weighs 291GB, while the Ethereum blockchain, almost 4 and a half years old (the genesis block is dated 30 July 2015), weighs almost double: 455GB (equivalent to downloading 350 high definition movies). See **https://bitinfocharts.com** for more info on blockchains data.

To address the blockchain bloat issue, in ZooBC, each node periodically (at a block height agreed by the network) takes a snapshot of the current state of its database, and computes a set of hashes for this snapshot. To be sure the node has the same snapshot as all the other nodes, it compares its new snapshot hashes against the hashes of the snapshot as calculated with the hashes in the metadata of a new spine block proposed by a blocksmith.

If the blocksmith uses hashes that, combined with known snapshot hashes, lead to the same set of hashes the majority of nodes in the node registry has calculated, its block is approved; otherwise it is rejected. Any new node joining the network then only needs to download the spine blocks until it finds the block with the hashes of the latest database snapshot, and downloads it in chunks from its peers, to come up to a recent database state, from which it can finish downloading the most recent blocks to catch up to the rest of the network.

The maximum size of a completely new snapshot can be determined by the sum of all the assets and accounts with current balance and relative properties. To produce a valid number in Mb we need to wait for the beta version to be running and do stress tests to the blockchain.

VIDEO

How Snapshot Works in ZooBC

What ZooBC Blocks Store the Snapshots?

What Is Snapshot Time Interval?

Relation between Spine Blocks and Snapshots

FORUM

Join discussions about Snapshot

ZOOBC Q&A

Get your questions about Snapshot answered

Creating Snapshots

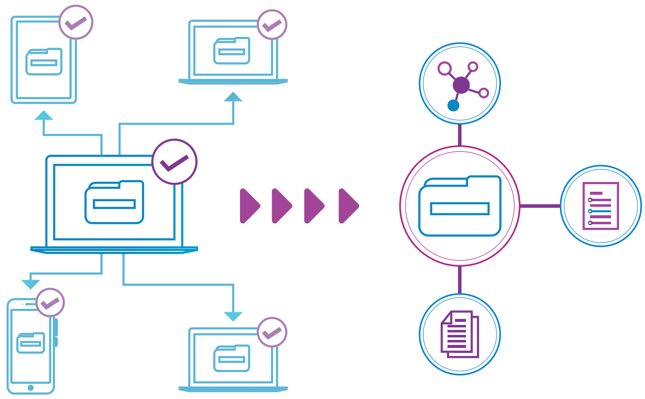
Approximately once per month, all nodes deterministically select a block height at which the next database snapshot should be taken. Each node will wait until enough blocks have elapsed that they have surpassed the "maximum rollback height", in order to guard against instances where the node begins computing the snapshot and then must abandon the process to process a rollback (**see glossary in Appendix 3 at Page 116**).

At this point the node initiates a background process to begin constructing a file that represents the exact state of the node's database from the earlier block height determined to be the snapshot height. Once this file is constructed, it is saved as a snapshot, and its hashes are computed.

In order to give nodes time to construct the snapshot file, a grace period of many more blocks is given before it becomes legal to include the new snapshot hash into a spine block. This ensures that once the new spine block is broadcast, all nodes, even those running on very limited hardware such as an Arduino or Raspberry Pi, should be able to construct and hash their snapshot file, and thereby validate the hashes in the new spine block against the one they independently computed.

Block Backups

State snapshots allow a node to zoom to the current state of major intervals in the blockchain's history. However, in order to rebuild the database state at any particular block height, a record must still be kept of all previous blocks and transactions. Additionally, in some cases the information in a transaction itself (not the resulting database state) may be required by a user, such as a digitally signed message.



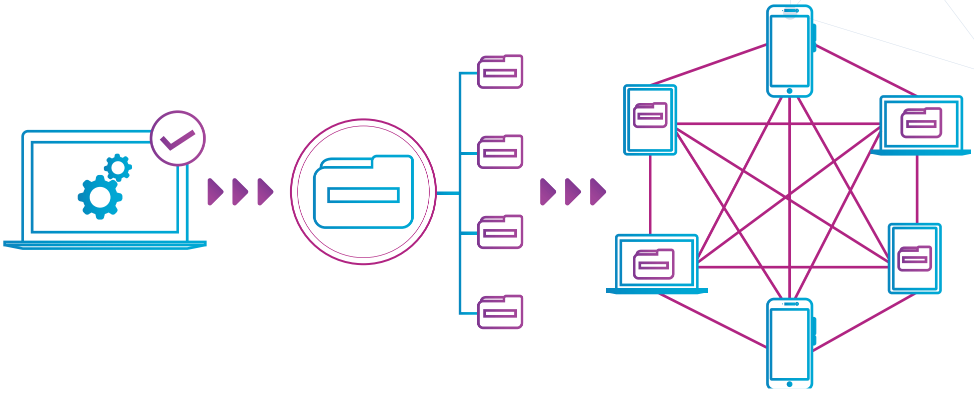
In the same way that a node produces large snapshot files, it will also periodically produce files which contain sets of blocks, transactions, and receipts, from a particular range of block heights, that needs to be archived. Another node that wishes to inspect or replay these blocks after they have been archived and pruned from its own database may request from other nodes to download the needed blocks backup file.

If we assume all nodes keep all block backups, this may seem like an equivalent (or worse) strategy to having the node simply store all historical blocks in the blockchain. But combined with a general mechanism for sharding the storage of large files across nodes on the network (described below), this allows for major space savings.

FORUM

Join discussions about Block Backups

File Distribution



It is important that the network remembers old state snapshots and block backups in a decentralized way, both because new nodes need to catch up to the network without a central point of truth, and because a node operator may want to go backward in time and validate an old transaction in the context in which it happened, or recover other states which only existed at a particular time.

Every node keeps the full data of the two latest snapshots in order to make them maximally available to new nodes joining the network. However, after this time, it is overly redundant for them to be duplicated across all nodes. Therefore we employ a strategy similar to the Torrent protocol to fairly divide the work of storing old snapshots and block backups.

Each file is subdivided into smaller chunks, and the responsibility for which set of nodes in the registry should retain each chunk is computed deterministically. As a network parameter we specify only the number of redundant copies of each chunk that the network should maintain, and the chunk assignment algorithm automatically updates this responsibility when nodes enter or leave the registry.

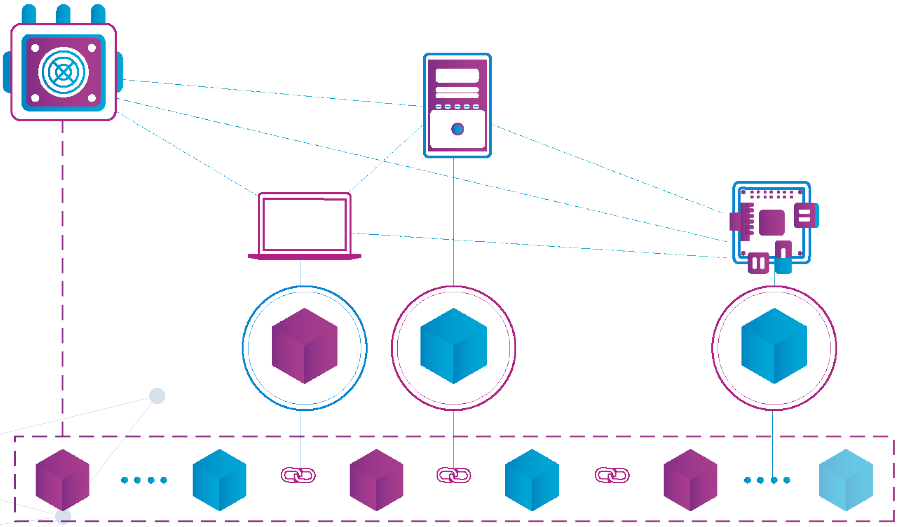
When a set of nodes is chosen to be the custodian of certain chunks of the files, there is no need for those chunks to be transmitted, meaning that other nodes can delete those chunks to free up space on their hard drives. These files are generated locally the same way by each node, and only need to be transmitted to another node upon request. This saves most network traffic compared to the Torrent protocol (on which ZooBC's decentralized storage is modeled), where a single peer posts a file and other peers download chunks from it to keep redundancy of the file in the network.

When a node wishes to retrieve a large file such as a snapshot from the network, it will first query other nodes with the hash of that file for a manifest of the hashes of the chunks which compose the file. For each chunk hash, the node can compute which set of registered nodes are currently responsible for storing that chunk, and can pick one at random from which to download it. Once all the chunks have been collected, they can be assembled and hashed together, and the final hash can be verified against the requested file hash.

ZooBC computes the number of redundant copies of any chunk as the square root of the registry size. In this way, as the network grows, more redundant copies of each chunk are maintained, but the number of copies grows more slowly than the size of the network, therefore the burden of each node on average continues to be reduced (adding nodes to the network reduces the storage responsibility of any given node.)

Archival Nodes

When users run ZooBC nodes, they can do so in the smallest devices, as ZooBC node doesn't have data bloating, doesn't require high computation to secure the blockchain, and thus can be run at low costs. Yet, if a user decides to keep a full copy of all the data present in the blockchain, they can set up the node in a stronger machine with a large data storage system, and set his node as an archival node. This means that its node will be one of those that guarantee full access to each piece of data that has transited in ZooBC since day 1. This can be done to run statistics on all the data, and to provide to the P2P network access to all the past data, when the assigned backup nodes are not available. Data can always be verified as good by recalculating its hash and matching it with the one in possession by the requestor.



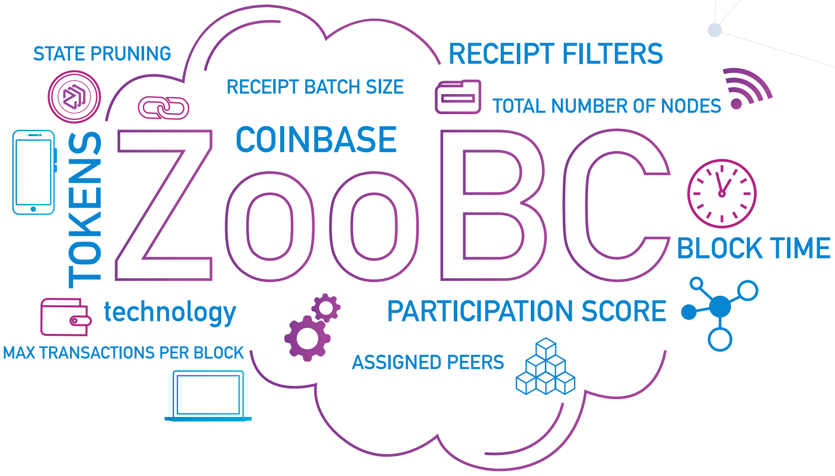
Although we leave open the possibility, via spine blocks, for casual nodes to accelerate to a recent database snapshot, never having previously downloaded or validated the chain, some node operators may wish to run a node which builds the entire chain history from scratch and maintain all data as a source for other nodes to download from.

To facilitate this we allow a node to be configured as an archival node. In this configuration the node will download and verify the entire history of the blockchain, and will retain all blocks, transactions, receipts and other data as a download source for other nodes.

This is not only a service to all the other nodes in the network, but may be a necessity for centralised services that need to run queries on the full history of the database, such as block explorer websites, or services to produce statistics or summaries of data in the blockchain. Any data provided by an archival node can be verified against the code ZooBC blockchain just by jumping to the needed transaction using the spine blocks as shortcut.

While ZooBC uses a distributed file storage strategy to enable any node on the network to recover past data from its regular peers, there is always the possibility of a catastrophic failure where all nodes maintaining redundant copies of some chunk go offline simultaneously. A few people maintaining archival nodes help mitigate the risk, as the network can then recover the lost pieces. There are many reasons for people to operate archival nodes without an explicit reward mechanism, such as operating blockchain explorer applications or other applications which need a complete historical index of the data.

Constants



While we have described in general terms the behavior of new systems and algorithms we have developed, many of the hard numbers and constants which will be deployed in the full release of ZooBC have not yet been determined. During our alpha and beta testing phases we will continue to reason about the best values for these parameters. Some of the major sets of constants are described below.

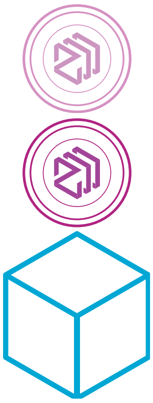
Tokens

The total amount of tokens that will be ever produced by ZooBC (token cap.) We will evaluate technical considerations like fungibility and psychological considerations to arrive at what we feel is the best total token supply.

https://blogchainzoo.com/glossary/z/zoo/

https://blogchainzoo.com/glossary/z/zoobit/

Coinbase



The rate at which new tokens are created by the network, the curve describing how this number will change with time, and how long it will take to generate and distribute them all. We aim to reward early network participants more than later ones to incentivize early participation, while ensuring the rewards will still be sufficient to incentivize node operators for decades.

Participation Score

The score earned or lost by a node when publishing receipts, finding blocks etc, and its default initial score when entering the node registry. This will influence how difficult it is for nodes to rise or fall in score, which consequently influences how easy it is to maximize your rewards or to be kicked from the registry.

VIDEO

Relation of Participation and Rewards

FORUM

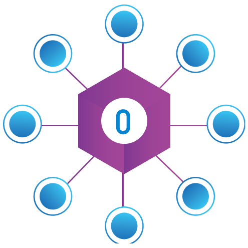
Join discussions about Participation Score

ZOOBC Q&A

Get your questions about Participation answered

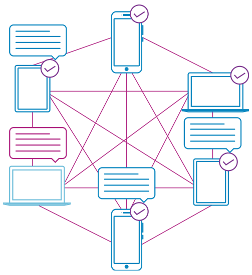
Get your questions about Rewards answered

Total Number of Nodes



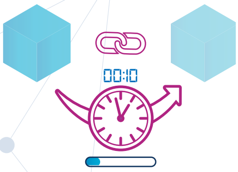
The number of nodes we aim to register in the genesis block of the network, along with the maximum registry size (if any) and the rules governing the rate at which nodes are added from the queue. We aim to allow new nodes to join as quickly as we find to be secure with the proof of participation algorithm.

Assigned Peers



The number of peers from the registry each node is assigned during each network topology period. Selection of this number will be based on the number of simultaneous open connections we can expect from any given node on the network, and how many maximum hops we want a transmission to take to be gossipped to all nodes in the network. Particularly we balance minimizing the hops for a piece of data to traverse the entire network with minimizing the number of simultaneous open connections we expect any given node to have.

Block Time



The average time between blocks. We aim to reduce this as much as we can without causing forking problems, as this will make the blockchain more responsive.

VIDEO

Block Timing Algorithm

ZOOBC Q&A

Get your questions about Block Time answered

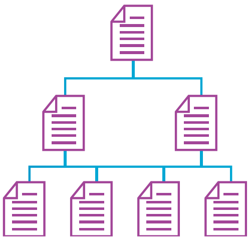
Max Transactions per Block

Closely related to block time, we aim to tune this number as high as we can without creating network problems, and without creating a centralizing requirement of high-powered computers to run nodes.

Receipt Filters

Receipt filtering involves several parameters, including how tightly to restrict the data filter, how to compute the expiry time from the network size, the number of assigned peers etc. If these filters are too restrictive, honest nodes will see their participation score fall unfairly, but if they are too wide, nodes will have room to skip participation or form attacking groups without being punished. Through testing we will refine each of these numbers until the network operates smoothly and securely.

Receipt Batch Size

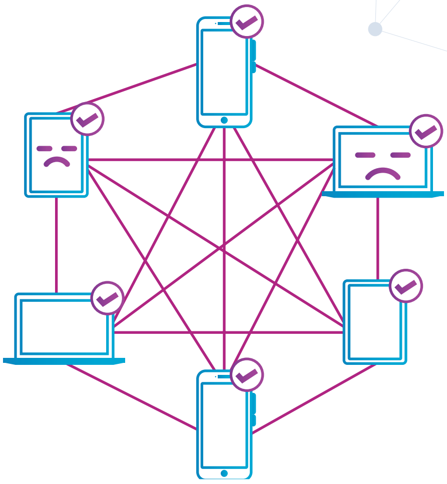


The number of receipts which are used to generate each Receipt Merkle Root. In tuning this parameter, we aim to maximize the number of receipts which can be proven by any given merkle root, while minimizing the time between a receipt being collected and being proven on the network.

State Pruning

Which pieces of data can safely be removed from state snapshots, and how long to retain them, and other old unused accounts, etc are removed, and if so when. How to calculate the fee for a piece of data to be posted so that it survives pruning for a long time. We aim to minimize the size of the state without losing any valuable information.

Attack Vectors



A decentralized technology is only as useful as its resistance to attacks from malicious actors. We have developed several theories of attack against the protocol we propose here, many of which have been mentioned in the sections to which they apply.

In this section, in a later version of this paper, we will list and analyse all possible attack vectors on ZooBC. Blockchain Zoo will hire white hat hackers to attempt attacks, as well as offer bounties for users demonstrating an attack on ZooBC. Additionally we will perform attacks against our own test networks and collect the results.

As we continue to refine the protocol and gather data from our alpha and beta testing phases, and collect feedback from the community, we will create a detailed accounting of possible attack vectors, how we have simulated them, and any measurements we have made demonstrating the security of the protocol.

ZooBC Tools

Wallet

VIDEO

How to Use the ZooBC Wallet

Wallet Features

What Makes the ZooBC Wallet Unique?

FORUM

Join discussions about 3rd Party Clones and Wallets

Join discussions about Wallets developed by ZooBC

ZOOBC Q&A

Get your questions about Wallets answered

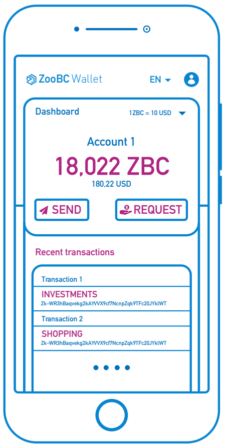
How to get some ZBC testnet tokens to play with ZooBC

At the moment we have decided not to publish a public testnet faucet. During the Alpha and Beta releases of ZooBC, there will be chain resets (the blockchain gets rebooted to accommodate changes to the protocol) leaving users suddenly with an empty wallet. To make it easier to trace who already had received testnet tokens and who got them before a chain reset, and to warn users of testnet chain resets, we are giving testnet tokens to the community support team.

Request ZBC testnet tokens via:

- Feedback function in the mobile or desktop wallet
- Forum - Token thread
- Telegram group

Mobile Wallet



ZooBC team is implementing Android and iOS mobile wallets structuring the code to make it easy for integrations and custom implementation. The mobile wallet includes a dApps section, where the UI for dApps can be loaded for seamless interaction of the user with the app. The mobile wallet integrates with hardware wallets allowing the user to secure private keys outside the mobile phone.

Download the iOS and Android Alpha version of ZooBC Mobile Wallet.

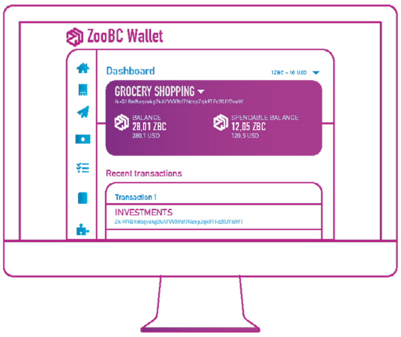
Participate in ZooBC Mobile Wallet Testing

The Android version is available during the Alpha and Beta versions of ZooBC. It is not available in the Play Store. The iOS Alpha version has been released with TestFlight (an iPhone app to test applications before they are put online in the App Store). At the moment of the release of the Beta version of ZooBC, in the TestNet, both Android and iOS versions will be published in the relative app stores.

Send us your feedback with ideas, suggestions, or bug reports.

Web Wallet

ZooBC also has a default web wallet to allow users to access their core account data and functionalities from the web. The web wallet allows nodes owners to interact with all the functionalities needed to manage and monitor blockchain nodes. Also the web wallet is being implemented to load specific dApps UI and to accept signatures made with hardware wallets and government released IDs.



Go to the ZooBC Desktop Wallet

VIDEO

Node Registration in the Web Wallet

How Can a User Register the Node?

Participate in ZooBC Desktop Wallet Testing

The Alpha Version of the web wallet is online and connects to the Alpha TestNet. Use either the mobile or the web wallet to generate your account address. Request demo tokens (testnet ZBC) to try the wallet using the feedback button. NOTE: many functionalities have not been implemented yet in the wallet, some aren't yet in the node app. Several "coming soon" functions show what will be available soon.

Send us your feedback with ideas, suggestions, or bug reports.

Key Management

Digital signatures, a core part of how a user interacts with a blockchain, requires the management of seed phrases and private keys. If a user loses or forgets its seed words or its private key, it loses access to its blockchain account and nobody can provide help to recover the account. Most web wallets offer the possibility to store an encrypted seed phrase in a centralized server, that the user can download and decrypt to restore his access to the account. Other web wallets offer multisig accounts, where 2 out of 3 signatures are necessary to execute transactions. The user holds two, and the web wallet holds one. This offers security as the web wallet cannot execute transactions alone, the user can, and the web wallet exists as a spare key in case the user loses one of the two it has. ZooBC is working to offer innovative key management to allow users to safely store their keys and restore access to their accounts when needed.

VIDEO

What Is the ZooBC Key Management Feature?

Block Explorer

Blockchain Zoo is working on a Block Explorer system, made of a server-side application and a web UI interface, to offer visualization of blockchain data from the basic ones such as blocks and transactions, to data specific to Proof of Participation protocol. This tool can be customized to highlight specific dApps state and be deployed by anyone that needs to offer to their users a specific frontend to monitor pieces of information specific to their application.

Explore the live blockchain here

FORUM

How to Install the Explorer on a MacOS

Join discussions about Explorer

Join discussions related to the explorer

ZOOBC Q&A

Get your questions about Explorer answered

Participate in ZooBC Explorer Testing

Like every blockchain, ZooBC also has a block explorer. It is a work in progress, but you can use the Alpha version of the block explorer to monitor the transactions you send and receive, and in the overall the transactions send in the blockchain. As ZooBC has a very unique way of working, due to the use of Proof of Participation, the block explorer also shows details that are specific to this protocol. If you have feedback or a question, if you have ideas on what the block explorer should visualize, if you feel something isn´t working properly... send us your feedback!

Explore the live blockchain here

Conclusion

As described in the introduction, this first version of ZooBC technology makes several extensions on earlier blockchain technology. We introduce the Proof of Participation mechanism, the mechanics of the node registry, state snapshots, and the spine blocks, a limited governance model for transaction fees, escrowed and liquid transactions, and many other innovations.

We have chosen our goals for this first release conservatively, especially emphasizing wide decentralization over scalability on the base layer, as we believe the censorship resistance and community ownership which come with such decentralization is the entire purpose of the technology. However this does not mean that we don't intend the technology to scale, or be contained by any of the conventional limits of blockchain technology.

Towards the Future...

ZooBC is not only a technology, but a research initiative. The knowledge we accumulate in our implementation and experience operating the network, and any funds we raise to achieve these goals, will be directed towards advancing our passionate vision to create decentralized technologies which will overcome the hurdles they face today in achieving widespread mainstream adoption. The exact details of this work still await our experience and design, but we can give a glimpse of the path ahead.

The node registry, especially, will serve as the backbone of a decentralized application (DApp) platform that Blockchain Zoo intends to develop in the next release of ZooBC. As each node's operator can choose which DApps to run (based on the node specifications), subsets of registered nodes will be able host the DApps they select, and thus be rewarded for operating that particular DApp. In this way the operator of a node in the registry not only stands to earn coinbase rewards from the greater network, but also has the opportunity to earn greater rewards by supporting projects or applications built on top of ZooBC.

The security and decentralization of each DApp depends on the number of nodes running such DApp. This will require DApp creators to include meaningful incentives in the logic of their decentralised applications, so that as many node operators as possible choose to run the DApp. This becomes a scalability solution in itself, a kind of “self-sharding”, where only subsets of the full network manage data and consensus for individual decentralised applications.

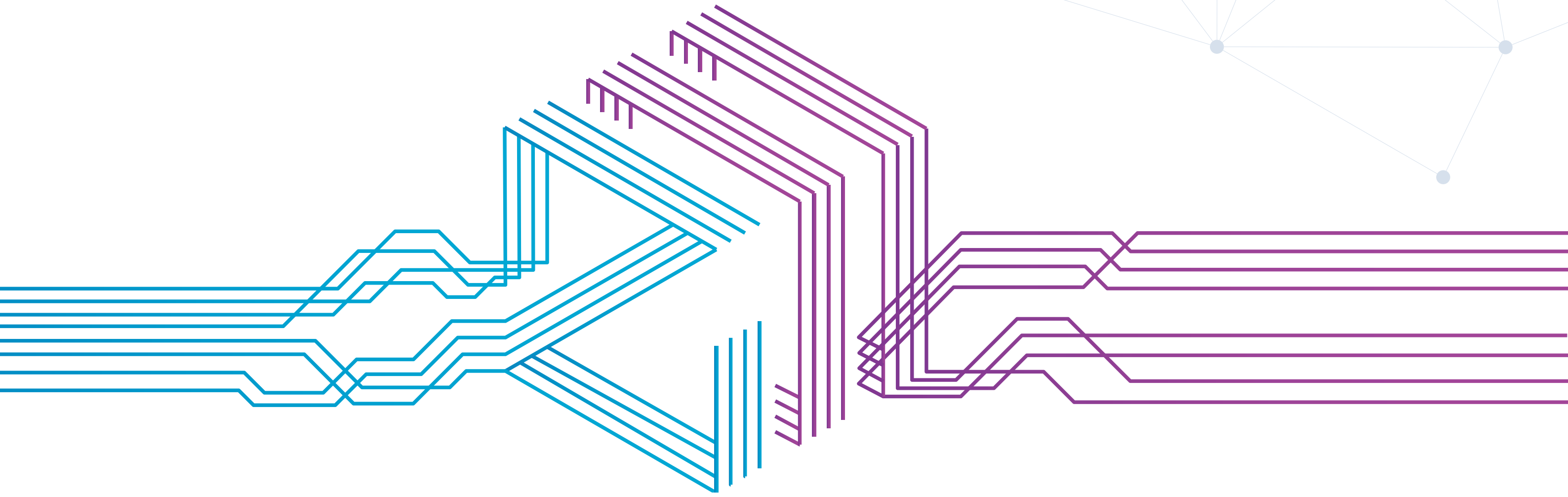
At the base layer, we will provide the tools for these DApps to accept digital signatures in multiple formats, and to give user accounts control over which transactions they agree to receive. We wish to facilitate a level of government compatibility not only on the base layer, but also for DApps built on the base layer, to create an ecosystem of applications which can be used to manage contracts, titles, and other "things" which may be demonstrated in a court of law.

In future versions, ZooBC will also adopt what has been developed in the field of zero-knowledge proofs, offering users and DApp developers the option to increase both the privacy and compressibility of user's transactions on the blockchain. This should be accomplished in such a way that it is compatible with the support for DApps, giving both users and DApp developers the option to use built-in zero-knowledge tools to easily enhance transactions within their own applications.

This is only a glimpse of where we would ultimately like to go. Blockchain Zoo looks forward to present, in future papers, detailed designs of these and other mechanisms in the next versions of the ZooBC technology and, with the support of the community, fully implemented technologies to the community.

All of us who used FidoNet, Torrent, and then Bitcoin witnessed the first sparks of a decentralized future, and many of us are now engaged in a community effort to fan those sparks into a flame which is transforming the entire information technology space. Blockchain Zoo, through its continuing efforts on the ZooBC project, seeks to attract the right minds, resources and experience, and organize them to push the cutting edge of decentralization, allowing all users to interact directly with each other and removing dependence on central intermediaries. This was the vision of Satoshi Nakamoto, and we proudly endeavor to carry forward the torch.

Learn more about ZooBC



ZooBC Alpha Version has launched! We are getting feedback and working on debugging the code while implementing the last functionalities that must be released when the beta version is ready. We are also working on updating this white paper (it is still missing several descriptions of ZooBC functionalities, and in some parts it isn't very clear for readers). Another difficult and long task we face, before releasing the beta version, is preparing all the code documentation. The goal is to prepare a "how-to" manual that will guide step by step any developer into implementing their version of the core node software, keeping it fully compatible with ZooBC protocol. To learn more visit ***zoobc.technology.***

Support ZooBC

Forum, Q/A, and Feedbacks

For its growth, adoption, and strength, ZooBC relies on its community. Blockchain Zoo (the company behind ZooBC) puts a great effort to support the ZooBC community. In Appendix 5, there are many links for social media, discussion groups, etc. Here we want to highlight the 3 key tools for the ZooBC community.

Please join our Blockchain Community

ZooBC Forum. The central place of the ZooBC community. Users, from the geekiest supporters to the less tech-savvy, earn a participation score and gain access as they use the forum. Here we manage bounties and reward the community to help us promote ZooBC, help code it, etc.

Read and join ZooBC forum

ZooBC Knowledge Base. Here anyone can ask questions about ZooBC and, once a supporter becomes an expert, can answer questions posted by other community members! Questions and answers gain a participation score. Yes, also ZooBC Q/A system has a participation score :) The best answers are selected, and we plan to give out monthly prizes to the top supporters!

Visit the ZooBC Q&A website

Bounties and rewards

At the moment of publishing this V1.1 of the whitepaper (April 2nd 2020) Blockchain Zoo has already invested, in ZooBC, the estimated amount of USD 1,048,055 (you can see a live increase of the amount invested in the introductory paragraph at https://ZooBC.com). We have a team of 55 people, fully devoted to the technical and the commercial sides of the project. As we go live with the Alpha version we welcome your support both in manpower and in financial donations that will be used for bounties open to the community itself.

See the forum!

Blockchain Zoo - The ZooBC Team

"A decentralized system is one where multiple parties make their own independent decisions"

To support ZooBC you can donate at the addresses as specified in ZooBC.com website

APPENDIX 1 - White Paper Older Versions

Version 1.0

Download the (clean) V1.0 of the white paper in PDF version here:

https://zoobc.com/ZooBC%20Whitepaper%20V1.0.pdf

The above V1.0 of the white paper in PDF format has a blockchain proof of existence record:

https://proofofexistence.com/detail/897842dea01bc7d128be2bee1798020029ed6ef22101cd345b895fc80ec15cac

Version 0.2

Download the (clean) V0.2 of the white paper in PDF version here:

https://zoobc.com/ZooBC%20Whitepaper%20Draft%20-%20V0.2.pdf

The above V0.2 of the white paper in PDF format has a blockchain proof of existence record:

https://proofofexistence.com/detail/8a3b56fd764998e98281506334188da469b70cf9c83968d11df1f2214f307e3f

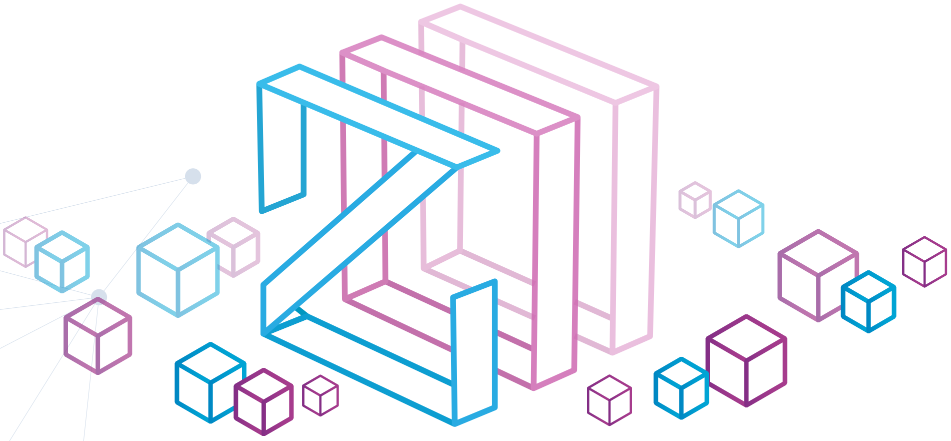
Version 0.1

Download the (clean) V0.1 of the white paper in PDF version here:

https://zoobc.com/ZooBC%20Whitepaper%20Draft%20-%20V0.1.pdf

The above V0.1 of the white paper in PDF format has a blockchain proof of existence record:

https://proofofexistence.com/detail/6a0cd36328a36cd8e744435a5b92890ba9134962f40f9165a4c15ff888fcad3e



APPENDIX 2 - Privacy Considerations

Moving into the future of Blockchain and decentralized systems, there is increasing concern over the entirely-public nature of the data on the blockchain. While these concerns are not addressed in ZooBC V1, they stand out clearly in our minds as important to address in the technology as a whole, and so here we list the thoughts that are guiding us as we begin designing V2 of the ZooBC blockchain technology.

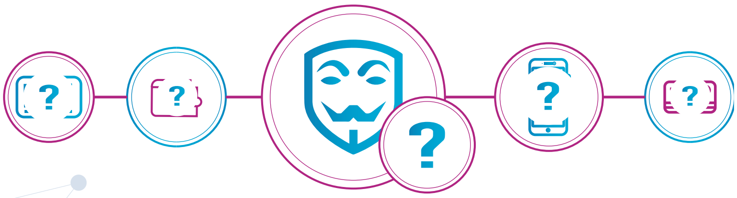
Data Privacy

ZooBC is implementing and will implement various aspects devoted to data privacy using zero-knowledge proof (ZKP) as a cryptographic method to allow a user (the prover) to prove to another user (the verifier) that they have the possession of some information without revealing the information to the verifier. This is important to keep the privacy of data that is exposed to the network and allow the use of decentralized applications when managing data that should not be revealed to the public. As ZooBC offers also distributed storage of data (initially used to store the blockchain and the chain snapshots, but later used also to save large data payload of blockchain transactions, and in subsets of nodes also decentralized apps datasets), the aspects of data privacy are a key factor for the adoption of ZooBC blockchain platform in industries that need to comply with regulatory requirements, and yet need to adopt decentralized systems to ease the business processes when working with suppliers and clients.

User Privacy

With the adoption of the **GDPR (General Data Protection Regulation)** in the **European Union,** questions regarding blockchain's compliance with the act have arisen. Personal data is "any information relating to an identified or identifiable natural person". Because identities on a blockchain are associated with an individual's public and private keys, this may fall under the category of personal data. A key part of the GDPR lies in a citizen's right to be forgotten, or data erasure. Due to the blockchain's nature of immutability, potential complications if an individual who made transactions on the blockchain requests their data to be deleted exist. ZooBC is also working to address this aspect of user privacy.

Anonymity



ZooBC aims to provide the needed tools for both systems and users that need full anonymity and applications and users that need or require defined identities for the users. Also, even for systems that need to identify users, there may be a need for anonymous transactions, anonymous voting, etc. ZooBC aims to use Zero-Knowledge-Proof transactions to allow anonymous transactions, anonymous interaction with decentralized apps, and most importantly, anonymous voting where a vote per registered user is guaranteed by the system, yet what the user has voted remains anonymous.

APPENDIX 3 - Glossary

This whitepaper uses some terms or expressions that require some background knowledge of blockchain. Here is a glossary to clarify some basic terms which may ease understanding of the whitepaper:

- **Node** (also referred to as "blockchain node", "network server", "peer"): A node is a computer or server that is running the ZooBC node software. More importantly, it is connected to other nodes, running the same software, to create a network.

https://blogchainzoo.com/glossary/n/node

- **Network** (also referred to as "P2P", "Peer to Peer", "the blockchain"): A peer-to-peer network that distributes computing tasks among many, private computers (decentralized servers), instead of using company computers (centralized servers).

https://blogchainzoo.com/glossary/p/peer-to-peer-p2p

- **Transaction** (also referred to as "payment", "transfer"): A transaction is a set of instructions that a blockchain user prepares and signs in a client application. The user then broadcasts the transaction to the network. Nodes in the network receive the transaction, execute it, and incorporate it with others into a block.

https://en.bitcoin.it/wiki/Transaction

Fee (also referred to as "transaction fee"): The payment a user grants to the network to include her transaction in the blockchain. To submit a transaction a user needs to add a fee that is given as an incentive to nodes to maintain the blockchain. The fee is usually attached to the transaction itself; if the transaction is rejected the fee is usually returned to the user. Fees also serve as a deterrent to users from spamming or otherwise abusing a blockchain. Quote: "Ah, if only spammers had to pay a fee to send us emails..."

https://blogchainzoo.com/glossary/t/transaction-fee

Mempool (also referred to as "queued transactions", "unconfirmed transactions"): A temporary cache of transactions stored by a node that will exist until each transaction is incorporated into the blockchain. When users submit a transaction to the P2P network, each node receives and validates the transaction and, before rebroadcasting it to the rest of the network, keeps a copy of it in a local queue called a "mempool". Once the transaction is included in a new "chained" block, the node first executes the transaction, updating the status of its copy of the blockchain, and then removes the transaction from its mempool.

https://blog.kaiko.com/an-in-depth-guide-into-how-the-mempool-works-c758b781c608

Block: A block (the word "block" in "block-chain") is a set of data, usually formed by metadata and transactions, assembled by a miner (see below) who ensures that it complies with the consensus algorithm of the blockchain. After creating a block, the miner signs it and broadcasts it to the network so that nodes can validate it and add it to the blockchain.

https://en.wikipedia.org/wiki/Blockchain#Blocks

https://blogchainzoo.com/glossary/b/block

Block Height (also referred to as "height", "blockchain height"): The position (height) of a block in the blockchain. A blockchain is composed of a sequence of unique blocks chained to one another, with each block given a sequential incremental value that determines its position in the blockchain. Each block is assigned a 'height' starting from zero. The blockchain height is the total number of blocks on a blockchain.

https://blogchainzoo.com/glossary/b/block-heigh

Miners (also referred as "forgers", "minters", "block creators", "blocksmiths"): The accounts, in a blockchain, that, through a node, create, sign, and propose new blocks to the network. Usually miners collect the fees of the transactions included in the blocks they create. In some blockchains they also collect an additional reward made of new coins created with the block (see "coinbase"). In Proof of Work an account can be owned by an individual user, or by a group of users pooling their computing resources (a 'mining pool') to increase their chances of earning fees.

https://blogchainzoo.com/glossary/m/miner

Coinbase (also referred to as "reward", "new coins", "new tokens"): The "sender" of a special transaction included in every new block (on some blockchains) that creates new tokens from nothing. These tokens are given as a reward to the miner that creates the new block, thereby increasing the total amount of tokens in circulation within the blockchain. Some blockchains use the coinbase only at block 0 and have 100% of their tokens already created when the blockchain is launched. ("Coinbase" should not be confused with the cryptocurrency exchange from San Francisco of the same name.)

https://en.bitcoin.it/wiki/Coinbase

Hashing Power (also referred to as "hashrate", "h/s", "work"): The total computing power used to calculate hashes in a Proof of Work blockchain. To make a new block valid to be broadcast to the network, miners of blockchains that use Proof of Work are required to find, through trial and error, a particular hash specific to their new block. To do so miners purchase specialized hardware that calculates millions of hashes per second. The quantity of hashes miners can collectively produce is called hashing power.

https://en.bitcoinwiki.org/wiki/Hashrate

https://blogchainzoo.com/glossary/h/hash-rate

Blockchain (also referred to as "chain", "distributed system"): A constantly growing list of data blocks, each containing users' transactions. Blocks are "chained" one to the other using cryptography. The "chaining" is done by adding to each new block a unique fingerprint (a cryptographic hash) of the previous block. How this unique fingerprint is generated is the core aspect of the mechanics that secure a blockchain (Proof of Work, Proof of Stake, or many other "Proof of..." that are used.)

https://blogchainzoo.com/glossary/b/blockchain

Public / Private; Permissionless / Permissioned: Different types of blockchain. A blockchain can be deployed publicly (openly accessible from the internet) or privately inside an access-protected Virtual Private Network (VPN). Blockchains can also be permissionless (where a user does not need to be "authorized" to submit transactions, run nodes, etc), or permissioned (where an administrator must authorize a user account before she can post transactions, etc.) When deploying a new blockchain, or a clone of an existing blockchain, the choice must be made to make the new blockchain public or private and permissioned or permissionless.

Public

https://blogchainzoo.com/glossary/p/public-blockchain/

Private

https://blogchainzoo.com/glossary/p/private-blockchain/

Permissioned

https://blogchainzoo.com/glossary/p/permissioned-ledger/

Permissionless

https://blogchainzoo.com/glossary/u/unpermissioned-ledger/

Fork (also referred to as "blockchain split", "alternative chain"): A fork occurs, like a fork in the road, when different sets of nodes in a blockchain disagree over which is a legitimate new block and create alternative versions of the chain. Most often a fork has a higher score and gets used by all nodes (see "Longest Chain", below.) A fork can be deliberate, or accidental. Between the deliberate ones there are "soft" forks and "hard" forks. A soft fork is a change in a blockchain protocol that is backward -compatible. That means that non-updated nodes are still able to process transactions and push new blocks to the blockchain, so long as they don't break the new protocol rules. A hard fork is a change in a blockchain protocol which is incompatible with the previous versions, meaning that nodes that don't update to the new version won't be able to process transactions or push new blocks to the blockchain.

https://en.wikipedia.org/wiki/Fork_(blockchain)

https://blogchainzoo.com/glossary/f/fork

Longest Chain (also referred to as "highest difficulty chain", "authoritative chain"): The chain of blocks that prevails in the case of a fork (see "Fork", above.) When miners "chain" a new block to the previous one, the way the blocks are "chained" has a value. The harder the work done in chaining a block (in PoW blockchains), or the closest a block complies to the consensus rules, the higher the score is for that piece of the chain. In the event of a fork, when a node has more than one new block or blockchain to choose from to update itself, the node calculates the cumulative "score" generated when creating each chain, and chooses the chain with the highest score (in Proof of Work this "score" is referred as "difficulty", and the chain with the highest cumulative difficulty is the chosen one.) In this way, nodes in a P2P network can reach the same decision without communicating with each other, but simply using the same algorithm to evaluate the options presented to them.

https://zdl-crypto.fandom.com/wiki/Longest_Chain

Proof of... (also referred to as "consensus algorithm/mechanism /model"): A set of rules to reach a consensus in a blockchain that allows nodes to "chain" blocks to one another by evaluating the validity of transactions and the "score" of new proposed blocks. In each blockchain nodes "prove" the validity of a transaction and a block using the same method, but many different blockchains can use different methods: the various consensus algorithms that are named "proof of..." followed by a single word describing the method (work, stake, capacity, etc)

https://blogchainzoo.com/glossary/c/consensus-algorithm

Receipt (specific to Proof of Participation in ZooBC): A node's acknowledgment of receiving data from another node, evidence of the latter's participation in the network. To measure its participation in ZooBC, when exchanging information in the Peer to Peer network, a node acknowledges having received information from another node by sending back a digitally signed receipt. Once a node has collected enough receipts, and when it is its turn to create a block, it can include in the metadata of the block a subset of the receipts it has collected. This can be later used to prove, at consensus level, that the node has participated in the network, thus earning a participation "score".

https://blogchainzoo.com/glossary/r/receipt

Token (also referred to as "coin", "crypto token", "cryptocurrency", "digital assets"): A unit of value within a blockchain system, at times used as an internal currency to pay for goods and services, but essentially needed to pay the transaction fees. The financial value of tokens is determined by their current market value which in turn depends on the level of user's trust in the blockchain.

https://blogchainzoo.com/glossary/t/token

Stake (also referred to as "locked funds", "total deposit", "locked balance"): A user's funds that are locked or held as a guarantee. Mostly referred to in "Proof of Stake" blockchain, the stake is the economic purpose to provably commit to a promise that the user won't sell the staked tokens for a pre-established period of time.

https://medium.com/coinmonks/understanding-proof-of-stake-the-nothing-at-stake-theory-1f0d71bc027

Address (also referred to as "account", "wallet"): An address is an alphanumeric string of letters and numbers that is unique to an "account". It is used to provide a digital identity (which can remain anonymous) to identify a sender or a recipient of blockchain transactions. For example to route digital assets across the network to a particular recipient.

https://blogchainzoo.com/glossary/a/address

Address Type (also referred to as "account type", "wallet type"): Addresses are the result of a particular mathematical algorithm. From a "private" key, which is secret to the user, a "public" key is calculated and from there the address the user can share with others. Government IDs and various blockchains have their own address format. While different blockchains have unique address formats, ZooBC supports many different types of addresses in a single blockchain.

https://unblock.net/what-is-a-blockchain-address

Digital Signature (also referred to as "signature", "cryptographic signature"): Digital signatures are a cryptographic tool to sign messages and verify message signatures in order to provide proof of authenticity for blockchain transactions.

https://blogchainzoo.com/glossary/d/digital-signature

Sybil attack: A Sybil attack is a kind of security threat on an online system where one person tries to take over the network by creating multiple accounts, nodes or computers. For example a Sybil attack can take place when somebody runs multiple nodes on a blockchain network. Attackers may be able to out-vote the honest nodes on the network if they create enough fake identities (or Sybil identities). They can then refuse to receive or transmit blocks, effectively blocking other users from a network.

https://en.wikipedia.org/wiki/Sybil_attack

Fee Scale (also referred to as "fee multiplier", "multiplier"): A variable number to which set fees needs to be multiplied by, to give an adjusted fee amount to be paid for transactions. Operators of registered nodes on the network may take a regular vote on the appropriate multiplier, which we call the fee scale, for minimum transaction fees. This guarantees that while the value of the blockchain token may fluctuate, the fees paid for transactions remain stable against the regular currency.

https://blogchainzoo.com/glossary/f/fee-scale

Peer Filter (also referred to as "connection table", "P2P topology"): The ordering of all nodes in the registry used to assign a set of preferred peers to each node. Each node computes its peer filter and it connects to the assigned nodes. When a node publishes receipts, the receipts it is allowed to publish must come from one of its assigned peers at the time the receipt was created.

https://blogchainzoo.com/glossary/p/peer-filter

Rollback (also referred to as "reorganization", "reorg"): The work a node does to replace the last blocks when it realizes to be in a fork of the blockchain as it receives a new chain that's longer (has higher cumulative difficulty) than its current active chain. Reorganizations happen when a node realizes that what it thought was the canonical chain turned out not to be. When this happens, the blocks in the latter part of its chain (i.e. the most recent transactions) are reverted and the transactions in the newer replaced blocks are executed. All reorgs have a "depth," which is the number of blocks that were replaced, and a "length," which is the number of new blocks that did the replacing.

https://learnmeabitcoin.com/guide/chain-reorganisation

Rollback Attack (also referred to as "51% attack", "Majority attack"): A rollback attack does not try to disrupt or interfere with the consensus protocol. Rather it plays along with the protocol's rules in order to attain the effect of changing the blockchain's content to the benefit of the attacker. If an attacker can create an authoritative fork with blocks excluding transactions she used to pay someone in the legit blockchain, will get all the other nodes to rollback (see above) into his forged version of the blockchain, where he never sent the payment, keeping its tokens.

https://learncryptography.com/cryptocurrency/51-attack

Merkle Tree (also referred to as "Hash Tree"): A Merkle tree is just an efficient way to prove that something is in a set, without having to store the set. Merkle trees are a fundamental part of blockchain technology. A Merkle tree is a structure that allows for efficient and secure verification of content in a large body of data. This structure helps verify the consistency and content of the data.

https://learncryptography.com/cryptocurrency/51-attack

Merkle Root (also referred to as "Merkle Proof"): The Merkle root is the hash of all the hashes of a set of data. In a blockchain block, all of the transaction hashes in the block are themselves hashed (sometimes several times - the exact process is complex), and the result is the Merkle root. The Merkle root, part of the block header, is the hash of all the hashes of all the transactions in the block.

https://blogchainzoo.com/glossary/m/merkle-root

Privacy (in Blockchain) (also referred to as "Zero-Knowledge Encryption"): Zero-Knowledge encryption means that service providers know nothing about the data stored on their servers. Zero-knowledge means that no one besides the user has the keys to her data, not even the service she is storing her files with. Also known as private encryption, it is the ultimate way in which a user can keep data private, though it does come with a few downsides: most important of these is that if the user loses the decryption key, the data is gone forever.

https://www.cloudwards.net/what-exactly-is-zero-knowledge-in-the-cloud-and-how-does-it-work

Zero-Knowledge Proof (also referred to as "Anonymous transactions"): A zero-knowledge proof (ZKP) is a cryptographic method which allows one person (the prover) to prove to another person (the verifier) that they have possession of some information without revealing the information to the verifier. Zero-knowledge proof (ZKP) private transaction protocol helps accelerate the adoption of secure, private transactions over public blockchains.

https://www.altoros.com/blog/zero-knowledge-proof-improving-privacy-for-a-blockchain

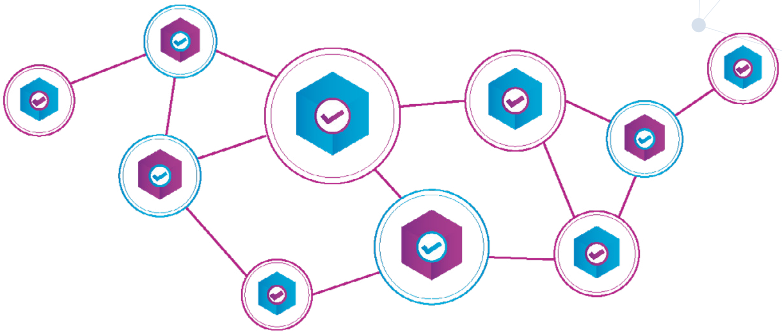
Digital Twins (also referred to as "Chain to Off-chain bridge"): "Digital twins" is the phrase used to describe a computerized (or digital) version of a physical asset and/or process. The digital twin contains one or more sensors that collect data to represent real-time information about the physical asset.

https://en.wikipedia.org/wiki/Digital_twin

Nonce (also referred to as "Salt"): A nonce is an abbreviation for "number only used once." In cryptography, a nonce is an arbitrary number that may only be used once. It is often a random or pseudo-random number issued in an authentication protocol to ensure that old communications cannot be reused in replay attacks. They can also be useful as initialization vectors and in the cryptographic hash function.

https://www.investopedia.com/terms/n/nonce.asp

APPENDIX 4 - Consensus Algorithms



The blockchain space is saturated with attempts to improve efficiency, security and fairness in the way that nodes reach a consensus on the history of events witnessed by the network. While the explosion of strategies may seem overwhelming or unnecessary, each project (some more than others) is doing its part in exploring the properties and tradeoffs yielded by each approach, and the crypto community is collectively narrowing down the proposed consensus strategies darwinistically until only the strongest are left standing.

Here a brief overview of the major approaches to blockchain consensus, and our reasoning to claim Blockchain Zoo's Proof of Participation as an improvement over its predecessors.

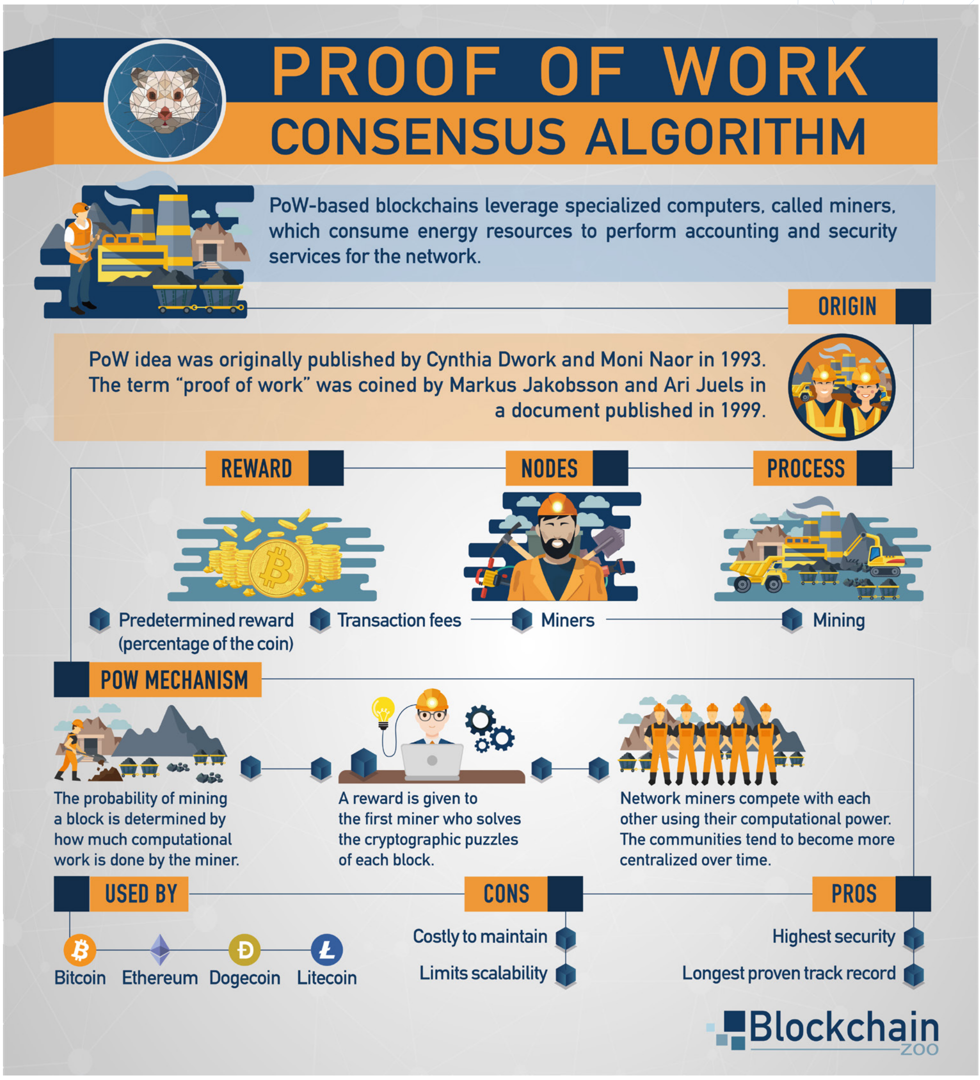
Proof of Work Consensus

The Bitcoin whitepaper introduced the concept of using accumulated "Proof of Work" as a method for any node to agree on which blockchain, among forks, should be trusted. This approach was very powerful because it allowed nodes to independently and objectively agree on one proposed history of events among many alternatives, in a way that resists a "Sybil attack" (because votes are counted by CPU cycles, not by accounts.) While many insist that Proof of Work is still the safest way to secure a blockchain, time has shown some undesirable properties of the algorithm.

First, the energy usage to secure such a system is always increasing, as miners participate in an arms race to claim more of the newly generated tokens. As of 2019, Bitcoin mining consumes as much energy as the nation of Switzerland (population 8.5 million), and as the token value appreciates the energy consumed is expected to rise. Some argue the security of this approach is worth the cost, but we believe this is a less-than-ideal property.

Second, this arms race has created a condition where an individual miner with average hardware is unlikely to find a block for himself during his natural lifespan. To fairly distribute the rewards for providing hash power to the blockchain, people have resorted to "pooling" their hash power together and proportionally dividing the block reward when any of them find a block. At the time of writing, there are about 12 mining pools with a non-negligible chance to successfully add blocks to the Bitcoin history, with all others unlikely to ever meaningfully participate. This level of centralization puts the "censorship resistance" property of Bitcoin in jeopardy, as it is not difficult to imagine a condition where a government or other organization may coerce 12 pool operators into complying with its demands to censor some transactions. We believe this failure to resist the tendency toward centralization is another less-than -ideal property.

Third, Proof of Work mining does not require the consent of any participants on the network for an outside party to become dominant. If someone truly had the money and the will to buy a majority of the hash power and use it to damage the chain, they could do so, even if all other long-time network participants wished to prevent it. It is debatable whether this is a desirable property or not, as it prevents the long-time stewards of the chain from ensuring they can maintain control for themselves. We believe it is fairer to give the vote to those who maintained and protected the network rather than to whoever has the money to accumulate hash power, and therefore we perceive this potential for externalized control to be a less-than-ideal property.



PROOF OF WORK
CONSENSUS ALGORITHM

PoW-based blockchains leverage specialized computers, called miners, which consume energy resources to perform accounting and security services for the network.

ORIGIN

PoW idea was originally published by Cynthia Dwork and Moni Naor in 1993. The term "proof of work" was coined by Markus Jakobsson and Ari Juels in a document published in 1999.

REWARD

- Predetermined reward (percentage of the coin)
- Transaction fees

NODES

- Miners

PROCESS

- Mining

POW MECHANISM

The probability of mining a block is determined by how much computational work is done by the miner.

A reward is given to the first miner who solves the cryptographic puzzles of each block.

Network miners compete with each other using their computational power. The communities tend to become more centralized over time.

USED BY

Bitcoin
Ethereum
Dogecoin
Litecoin

CONS

- Costly to maintain
- Limits scalability

PROS

- Highest security
- Longest proven track record

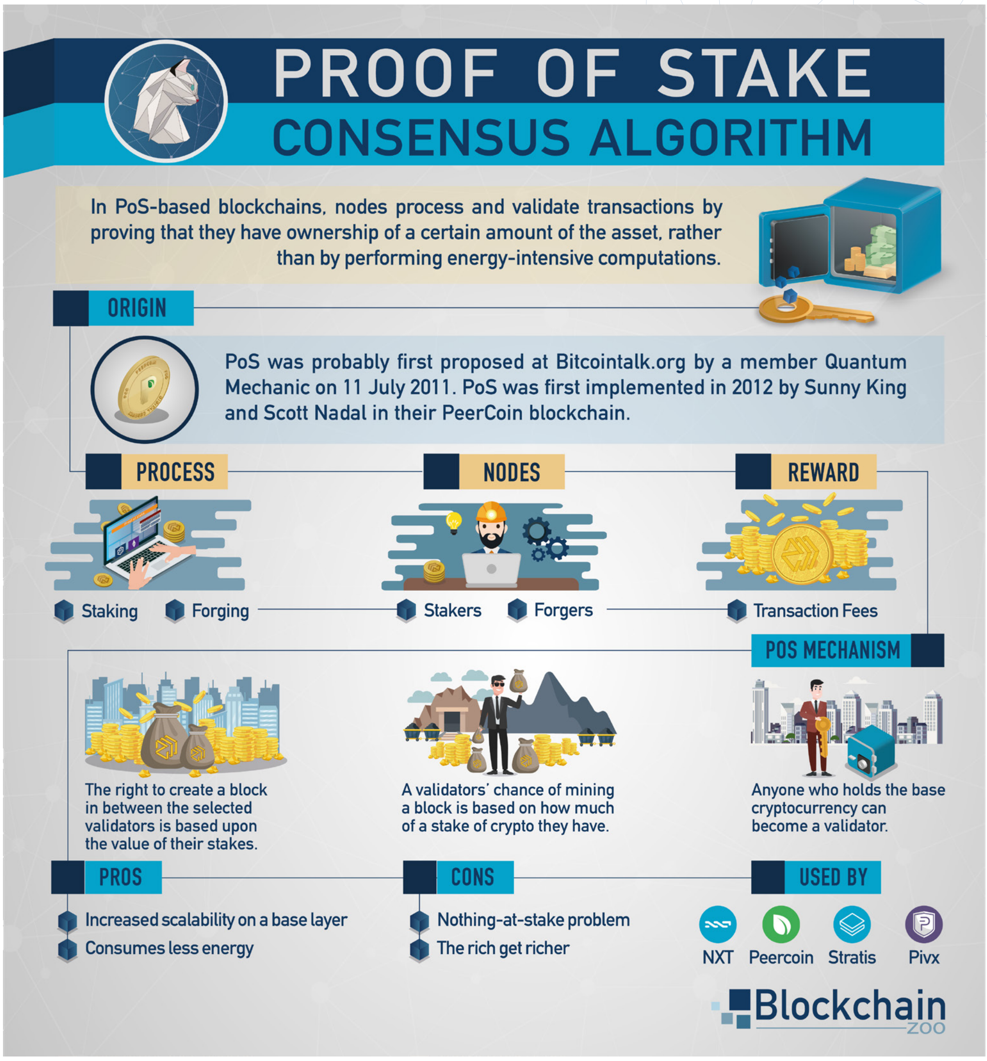
Proof of Stake Consensus

The first and third concerns described above motivated some to develop an alternate consensus algorithm to objectively choose between proposed versions of the blockchain history called "Proof of Stake". In this approach, the likelihood of a network participant to add a block to the history is computed according to how many tokens on the network she possesses, and the block she creates is proven to originate from her via a digital signature. In this way, which chain required "more work" to create is simulated by a calculation of which nodes added blocks at which times and their relative stakes. This design requires minimal energy and guarantees that, in a fork, the nodes will choose the blockchain created by the majority of highly-invested network participants -- in other words, those who have a larger stake of tokens locked to create new blocks. However, this strategy still has some undesirable properties.

First, the second concern described above regarding the centralization of mining power applies equally to Proof of Stake. When a new Proof of Stake blockchain launch, it has very often all the tokens that will ever exist "pre-mined". As a new blockchain is known by few, it is very likely that, either at the launch of the blockchain or at a later time, a majority of the tokens is in the hands of only a few participants that can distribute them in many anonymous accounts. These few participants will create almost all of the blockchain history, as well as claim any rewards such as the transaction fees, meant to incentivize the entire network to run nodes. If one party owns 51% of the stake, or parties who own as much decide to conspire, they can effectively censor the transactions which are accepted on the blockchain, or at a later time create an alternate history which will be accepted by the rest of the nodes.

Second, even in the event that the block-creating power is well-distributed, someone could come in possession of the private keys of accounts which, even if they are now empty, at some point in time had a large stake. Using those accounts an attacker can create an alternate blockchain history that starts from the time those accounts had a large balance. This alternative fork may be seen, by other nodes, as the most authoritative chain, forcing them to switch to it. An attacker might purchase past private keys for less than the potential gains of creating an alternate blockchain history. Similarly, if someone discovered a vulnerability in the node software which allows private keys to be copied from the servers, they could quietly collect enough of the participants' keys to take over the blockchain. Importantly, as Proof of Work advocates note, creating an alternate chain would require only trivial energy to be invested, and so this hijacking of a Proof of Work blockchain could be performed quickly and cheaply once the necessary keys were collected.

This second concern is exacerbated by the first: the more centralized the blockchain is, the more vulnerable it is to outside manipulation. In fact, if at any point in the blockchain's history the majority of the block-creating power was controlled by only a small number of private keys, anyone who can get their hands on these few keys, even if those accounts now are empty and abandoned, can effectively re-write the blockchain history from that time forward and have it accepted by other honest nodes running the network at the time of their choosing in the future.



PROOF OF STAKE CONSENSUS ALGORITHM

In PoS-based blockchains, nodes process and validate transactions by proving that they have ownership of a certain amount of the asset, rather than by performing energy-intensive computations.

ORIGIN

PoS was probably first proposed at Bitcointalk.org by a member Quantum Mechanic on 11 July 2011. PoS was first implemented in 2012 by Sunny King and Scott Nadal in their PeerCoin blockchain.

PROCESS

- Staking
- Forging

NODES

- Stakers
- Forgers

REWARD

- Transaction Fees

POS MECHANISM

The right to create a block in between the selected validators is based upon the value of their stakes.

A validators' chance of mining a block is based on how much of a stake of crypto they have.

Anyone who holds the base cryptocurrency can become a validator.

PROS

- Increased scalability on a base layer
- Consumes less energy

CONS

- Nothing-at-stake problem
- The rich get richer

USED BY

NXT, Peercoin, Stratis, Pivx

Blockchain ZOO

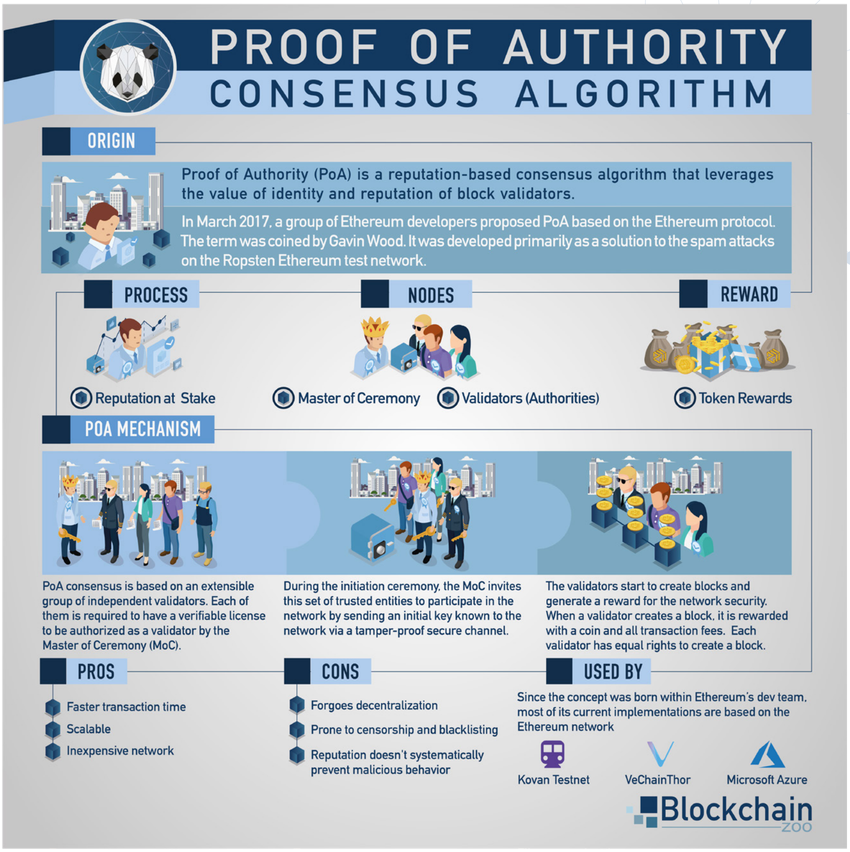
Federated Consensus

The first concern above can be addressed by having a fixed number of participants which each contributes to the blockchain history equally (regardless of hash power or stake), thus ensuring that network rewards and history-creating power cannot centralize to one entity. Further, the second concern can be mitigated by ensuring that the number of these participants (and therefore the number of keys which would need to conspire to create a new longest chain) is large. Consensus algorithms that embrace this theory of security are known as "Federated" consensus and have been well-studied long before the emergence of blockchain technology for use in other distributed systems. While we feel such strategies effectively address the concerns above, in other ways they are a step backward from Proof of Work and Proof of Stake.

First, federated networks are no longer "permissionless": It is no longer open for any person to simply join the network and begin participating in the consensus process, regardless of their merit or investment in the network. Usually some centralized process exists to govern which participants are admitted into or ejected from the federation, and this central process becomes precisely the weakness that must be avoided in truly decentralized systems, as it gives disproportionate control of the network to whoever manages the process. Some federated consensus models allow users to vote on who can be admitted into the federation, but this voting procedure may also be vulnerable to manipulation.

Second, this set of federated entities is usually well known (in fact Federated consensus strategies gain their promise of security by showing that the participants are separate well-trusted entities). This makes it easy for an external adversary to identify who needs to be pressured (technically, legally, financially or otherwise) to censor the network. Additionally, by virtue of the fact that the participants are already connected by at least one entity (the entity which approves their membership in the Federation), it is not difficult to imagine them conspiring off-chain to achieve some mutual goal.

For these reasons we feel a pure federation is not acceptable for secure decentralized consensus, although it has some properties we would like to preserve. Proof of Authority and Proof of Reputation (based on Proof of Authority) are two examples of federated consensus algorithms.



PROOF OF AUTHORITY CONSENSUS ALGORITHM

ORIGIN

Proof of Authority (PoA) is a reputation-based consensus algorithm that leverages the value of identity and reputation of block validators.

In March 2017, a group of Ethereum developers proposed PoA based on the Ethereum protocol. The term was coined by Gavin Wood. It was developed primarily as a solution to the spam attacks on the Ropsten Ethereum test network.

PROCESS

- Reputation at Stake

NODES

- Master of Ceremony
- Validators (Authorities)

REWARD

- Token Rewards

POA MECHANISM

PoA consensus is based on an extensible group of independent validators. Each of them is required to have a verifiable license to be authorized as a validator by the Master of Ceremony (MoC).

During the initiation ceremony, the MoC invites this set of trusted entities to participate in the network by sending an initial key known to the network via a tamper-proof secure channel.

The validators start to create blocks and generate a reward for the network security. When a validator creates a block, it is rewarded with a coin and all transaction fees. Each validator has equal rights to create a block.

PROS

- Faster transaction time
- Scalable
- Inexpensive network

CONS

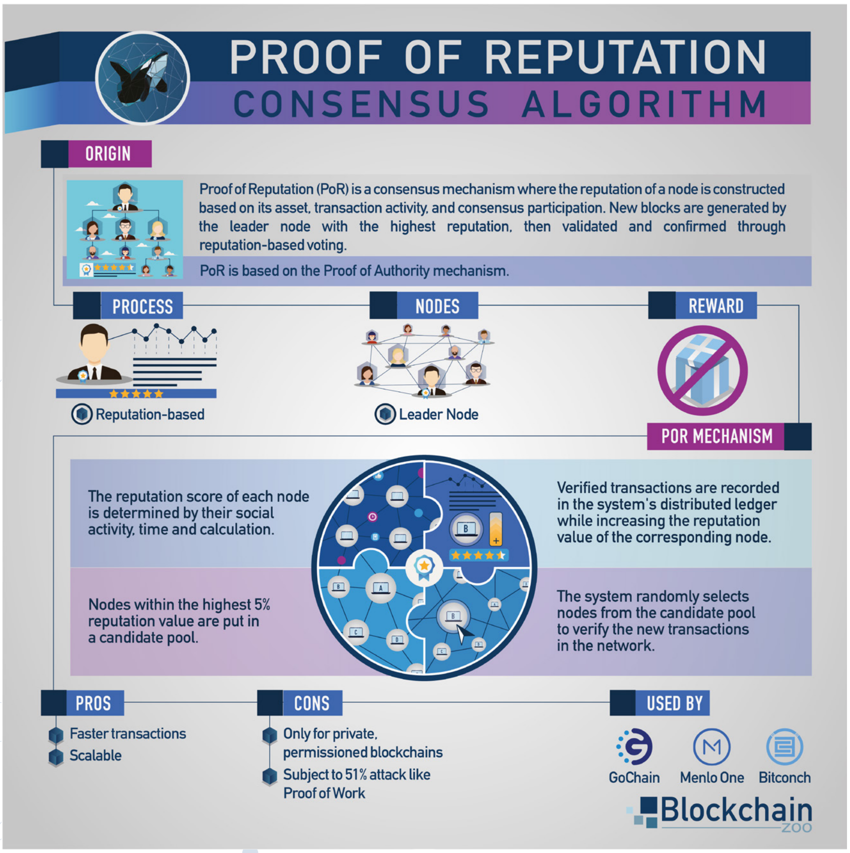
- Forgoes decentralization
- Prone to censorship and blacklisting
- Reputation doesn't systematically prevent malicious behavior

USED BY

Since the concept was born within Ethereum's dev team, most of its current implementations are based on the Ethereum network

Kovan Testnet   VeChainThor   Microsoft Azure

Blockchain ZOO



PROOF OF REPUTATION CONSENSUS ALGORITHM

ORIGIN

Proof of Reputation (PoR) is a consensus mechanism where the reputation of a node is constructed based on its asset, transaction activity, and consensus participation. New blocks are generated by the leader node with the highest reputation, then validated and confirmed through reputation-based voting.

PoR is based on the Proof of Authority mechanism.

PROCESS

- Reputation-based

NODES

- Leader Node

REWARD

POR MECHANISM

The reputation score of each node is determined by their social activity, time and calculation.

Verified transactions are recorded in the system's distributed ledger while increasing the reputation value of the corresponding node.

Nodes within the highest 5% reputation value are put in a candidate pool.

The system randomly selects nodes from the candidate pool to verify the new transactions in the network.

PROS

- Faster transactions
- Scalable

CONS

- Only for private, permissioned blockchains
- Subject to 51% attack like Proof of Work

USED BY

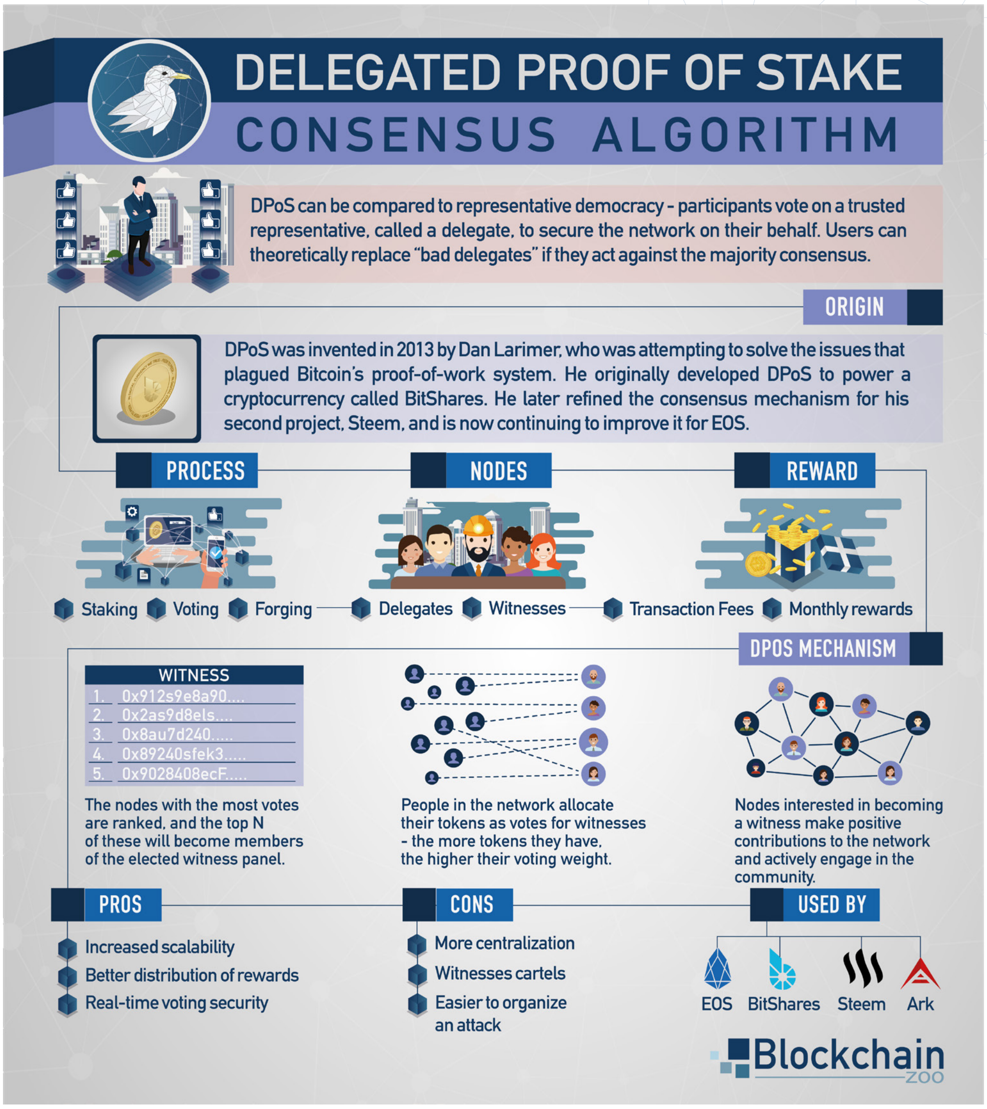
GoChain   Menlo One   Bitconch

Blockchain ZOO

Delegated Proof of Stake Consensus

One of the most popular modern approaches to improving the scale of a blockchain network is to use "Delegated Proof of Stake" Consensus, where the accounts of the blockchain vote, with their stake, a small number of nodes, running large enough hardware, to become block creators and thus support a high transaction volume blockchain. While this approach can dramatically increase the throughput of the network, it does so at the expense of decentralization, having similar flaws as conventional Proof of Stake and small Federations. As described above, specifically, the ease of quickly collecting enough stake to control the network, and the ability of a small number of block creators to conspire to censor transactions.

While recognizing the importance of creating more scalable blockchain technology, a pseudo-centralized approach is not the correct path. ZooBC aims for the technical and financial requirements of operating a node to not exceed those available to the average user, enabling a blockchain that is directly operated by a large number of small, independent actors. ZooBC offers a solution which both vastly increases the number of keys which would need to be compromised to rewrite the blockchain, and makes it significantly more difficult for such actors to conspire or be coerced by an adversary seeking to censor the blockchain.



DELEGATED PROOF OF STAKE CONSENSUS ALGORITHM

DPoS can be compared to representative democracy - participants vote on a trusted representative, called a delegate, to secure the network on their behalf. Users can theoretically replace "bad delegates" if they act against the majority consensus.

ORIGIN

DPoS was invented in 2013 by Dan Larimer, who was attempting to solve the issues that plagued Bitcoin's proof-of-work system. He originally developed DPoS to power a cryptocurrency called BitShares. He later refined the consensus mechanism for his second project, Steem, and is now continuing to improve it for EOS.

PROCESS

- Staking
- Voting
- Forging

NODES

- Delegates
- Witnesses

REWARD

- Transaction Fees
- Monthly rewards

DPOS MECHANISM

WITNESS	
1.	0x912s9e8a90....
2.	0x2as9d8els....
3.	0x8au7d240.....
4.	0x89240sfek3.....
5.	0x9028408ecF.....

The nodes with the most votes are ranked, and the top N of these will become members of the elected witness panel.

People in the network allocate their tokens as votes for witnesses - the more tokens they have, the higher their voting weight.

Nodes interested in becoming a witness make positive contributions to the network and actively engage in the community.

PROS

- Increased scalability
- Better distribution of rewards
- Real-time voting security

CONS

- More centralization
- Witnesses cartels
- Easier to organize an attack

USED BY

EOS BitShares Steem Ark

Blockchain ZOO

Byzantine Fault Tolerant Consensus

Another popular strategy for increasing the transaction throughput of a decentralized network is an algorithm called "Practical Byzantine Fault Tolerance". This algorithm is especially used in Federated consensus, where the participants are pre-selected, because it carries a particular weakness in the face of Sybil attacks (when one attacker can operate many nodes on the network) which would make it unsuitable for some pBFT networks.

While the algorithm used can increase the speed of transaction processing, pBFT consensus suffers from a particular property that cannot be tolerated: an attacker controlling as few as 1/3 (one third) of the nodes can prevent the entire network from reaching consensus. This may be a safe assumption to make in a federated network with tightly regulated members, but in a public, permissionless network, we feel this is too vulnerable to attack. For this reason ZooBC follows in the tradition of blockchains where an attacker must reliably control more than 1/2 (half) of the network (and therefore properly be the majority) in order to have a chance to control the network's behavior.

https://medium.com/codechain/why-n-3f-1-in-the-byzantine-fault-tolerance-system-c3ca6bab8fe9

Proof of Participation Consensus

Based on consideration of the various flaws and tradeoffs in the consensus mechanisms explored above, ZooBC adopts a few elements of Proof of Stake and of Federated consensus strategies, combined with a novel algorithm developed by Blockchain Zoo to prove that a node is performing useful work for the network. We call this "Proof of Participation" consensus.

ZooBC maintains a federation of nodes that we call the "Node Registry". Only nodes within the registry are permitted to create blocks, and their probability to create the next block is more or less equal. This is similar to Federated Consensus. However, any node operator can apply for a spot in this registry, and their admittance into the registry is governed entirely by the protocol rules, not by any centralized entity. The rate at which new nodes are added to the registry is strictly limited by the protocol, and the selection of which applicants will be added is governed by how much stake they are willing to lock while they are in the registry. As nodes queue to enter the node registry, priority is given to nodes with a higher locked stake. This method uses a concept of Proof of Stake, to the extent that staking tokens (a scarce resource on the network) is used as a Sybil prevention mechanism, essential for a new blockchain.

NOTE: *The strategy of requiring nodes to lock tokens to join the "Node Registry" is applied only in the first version of ZooBC for the following reasons:*

1. *To give a use to the token;*
2. *To create demand for tokens;*
3. *To reduce the tokens in circulation, thus creating scarcity;*
4. *To limit the initial growth of the node registry;*
5. *To prevent Sybil attacks to the newly launched blockchain*

Future versions of ZooBC will shift from using "stake" for prioritizing access to the node registry, to participation score. The algorithm will prioritize nodes that, while in the queue, collect more participation score compared to others. This will remove the use of "stake" from ZooBC, making it a blockchain secured exclusively by participation.

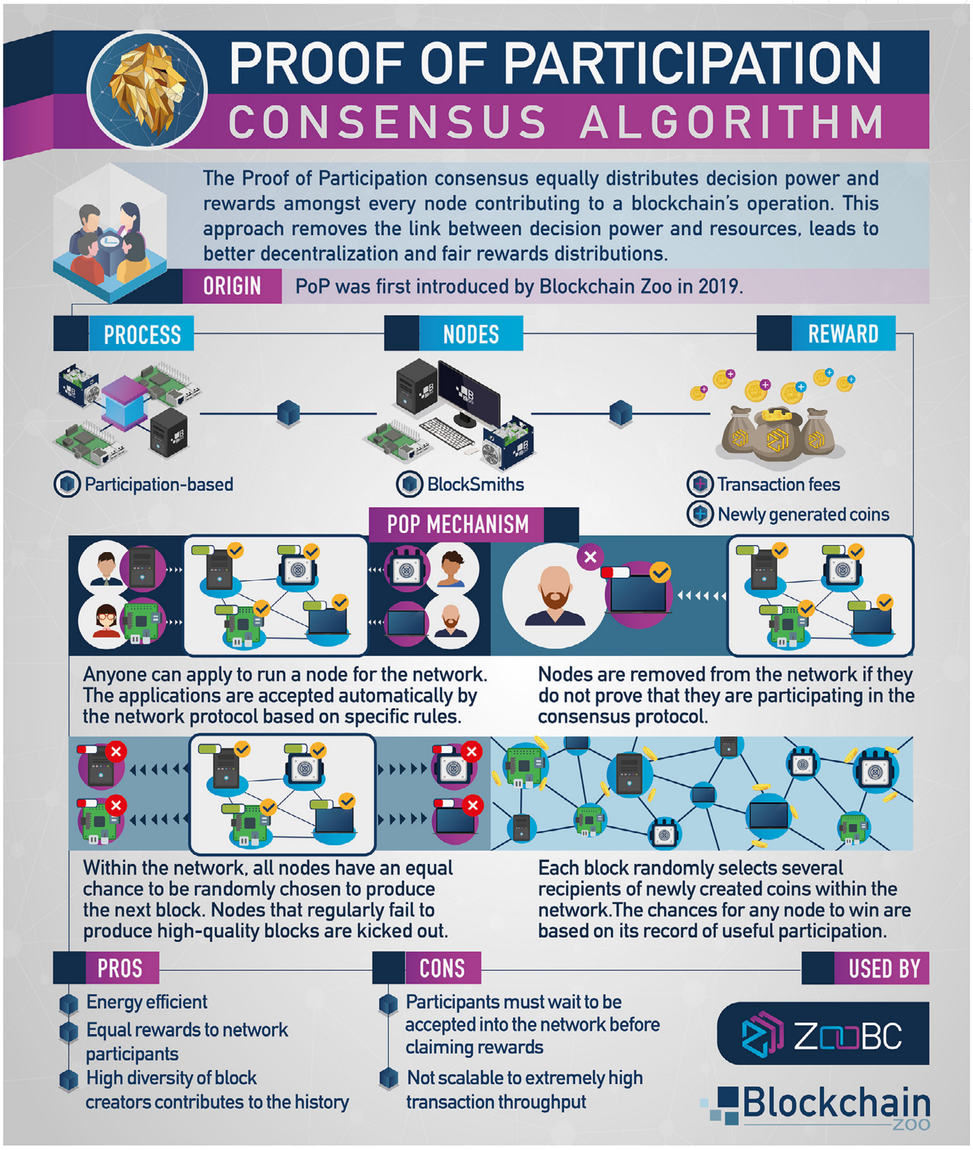
Finally, when nodes in the "Node Registry" produce a block, they must include in the block some proof that they have been contributing to the network (by honestly propagating transactions and blocks.) Nodes that miss their opportunity to create blocks, or which fail to include such proofs in their blocks, will gradually lose "participation score" until they are automatically removed from the registry. In this way, an operator must maintain a node that is in regular communication with the rest of the network to continue creating blocks and collecting coinbase rewards.

Coinbase rewards: as for Bitcoin, ZooBC has no "pre-mined" tokens. At the beginning of the blockchain the total amount of tokens in existence is zero. At each block new tokens are created and distributed to the nodes in the registry. When a new block is created, based on its block seed, a pseudo-random selection of nodes will receive new tokens. The new tokens created in the block are evenly distributed to the selected winners, but a node's probability of being selected as a winner is directly proportional to its participation score. In this way, nodes are strongly incentivized to maximize their participation score, as this also maximizes their profit on average. After enough time, all nodes which are participating reliably should reach the maximum participation score, making the distribution of new rewards between them essentially equal.

This is an element of fairness in rewards that most other blockchain technologies have not attempted to attain. For example, in Bitcoin, so long as you produce the block with the most work, no one can say whether you have been participating in other regular network activity. It is largely taken on faith that node operators are propagating blocks and transactions. In practice this leaves much of the hard work of decentralizing the network to enthusiasts who run nodes because they care, despite their not having enough hash power or stake to ever earn rewards from the network.

This strategy resolves many of the flaws described in the previous algorithms. By virtue of using digital signatures, ZooBC avoids Proof of Work's energy consumption problem. By the use of an ever-growing federation of nodes, ZooBC equalizes the probability of each node operator to add to the history and claim coinbase rewards and avoid the miner centralization problem of both Proof of Work and Proof of Stake. By requiring each node to prove its participation (in the form of digitally signed messages from other nodes) ZooBC dramatically increases the number of private keys which would be needed to be compromised for an attacker to forge a longer chain, mitigating the key-stealing weakness of both Proof of Stake and Federated consensus. By allowing anyone to freely apply for a spot in the node registry, ZooBC avoids the permissioned (and therefore centralized) nature of a fully-Federated network.

In the long term, this Proof of Participation strategy results in a very large pool of nodes taking turns to contribute to the network history, and being equally rewarded for their service. Creating a longer chain does not only require the keys of the majority stakeholders, but also of a majority of registered nodes in the network. To attack a PoP chain, an attacker needs to control far more than half of the nodes in the registry. Coming to possess a large majority of the registry not only requires a large investment, but also requires the time needed to have many new nodes gradually admitted into the registry, and carries the cost of running real nodes proving their service to the network for the entire duration of the attack



PROOF OF PARTICIPATION CONSENSUS ALGORITHM

The Proof of Participation consensus equally distributes decision power and rewards amongst every node contributing to a blockchain's operation. This approach removes the link between decision power and resources, leads to better decentralization and fair rewards distributions.

ORIGIN PoP was first introduced by Blockchain Zoo in 2019.

PROCESS

- Participation-based

NODES

- BlockSmiths

REWARD

- Transaction fees
- Newly generated coins

POP MECHANISM

Anyone can apply to run a node for the network. The applications are accepted automatically by the network protocol based on specific rules.

Nodes are removed from the network if they do not prove that they are participating in the consensus protocol.

Within the network, all nodes have an equal chance to be randomly chosen to produce the next block. Nodes that regularly fail to produce high-quality blocks are kicked out.

Each block randomly selects several recipients of newly created coins within the network.The chances for any node to win are based on its record of useful participation.

PROS

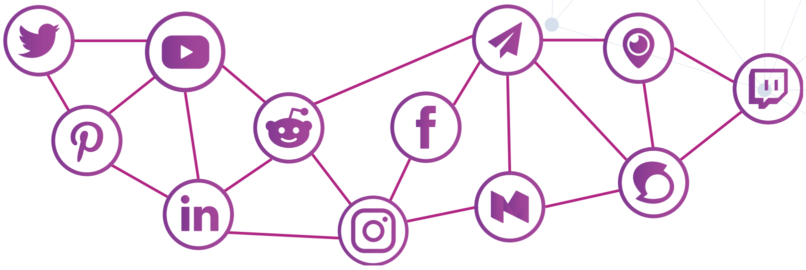
- Energy efficient
- Equal rewards to network participants
- High diversity of block creators contributes to the history

CONS

- Participants must wait to be accepted into the network before claiming rewards
- Not scalable to extremely high transaction throughput

USED BY

APPENDIX 5 - Websites, Groups and Social Media



You can contribute to the discussion, ask questions, and learn more about Blockchain Zoo, ZooBC Blockchain and BlockCoWork by visiting our websites, groups and social media channels.

Website

- ZooBC website
- ZooBCTechnology
- ZooBC Community Forum
- ZooBC Questions and Answers
- ZooBC Mobile App
- ZooBC Web Wallet
- ZooBC Block Explorer
- Blockchain Zoo website
- Blockchain Zoo & ZooBC Blog
- BlockCoWork website
- Blockchain Zoo Store

Join our community & groups

- Join our blockchain community
- ZooBC Facebook group
- ZooBC LinkedIn group
- ZooBC Telegram group
- Blockchain Zoo Facebook group
- Blockchain Zoo Telegram group

Follow us on social media

ZooBC

- ZooBC & Blockchain Zoo YouTube
- ZooBC Facebook page
- ZooBC LinkedIn
- ZooBC Twitter
- ZooBC Instagram
- ZooBC SubReddit
- ZooBC Telegram channel

Blockchain Zoo

- Blockchain Zoo Facebook
- Blockchain Zoo LinkedIn
- Blockchain Zoo Twitter
- Blockchain Zoo Instagram
- Blockchain Zoo Pinterest
- Blockchain Zoo Medium
- Blockchain Zoo Periscope
- Blockchain Zoo Twitch
- Blockchain Zoo Steemit
- Blockchain Zoo Telegram channel

BlockCoWork

- BlockCoWork Facebook
- BlockCoWork Twitter
- BlockCoWork LinkedIn
- BlockCoWork Instagram

APPENDIX 6 - White Paper Index

APPENDIX 6 - White Paper Index

ZooBC White Paper

New Strategies in Blockchain

Developed by

Blockchain
zoo

www.blockchainzoo.com