Proof of Justice’s Paper, Version 1.0.0.

Revolutionary new consensus algorithm maximizing speed, security, decentralization, scalability, and eco-friendliness.

Table of Contents

Introduction....4
  Prior Art & Issues....4
    Proof of Work....4
    Proof of Stake....5
    Proof of Space....5
  Proof of Justice – Overview....5
Base Architecture....6
  Hooks....6
  Node....6
      Mempool....6
  Transaction....7
  Block....8
    Transactions....8
    Peterson(s)....8
Scoring System....8
  Score....8
    Role Granting....9
  Levels....11
    Role-Dependent Levels....12
    Possible Drawbacks....12
    Preventing Exploits....13
  Resource Access Granting....13
  Entity Reporting System....13
  Types of Rules....14
    Regular....14
    Non-deterministic....14
    Requirement....15
    Score-Dependent....16
    Adaptive....17
    Action-Dependent....17
    Long-Term Memory....17
    Short-Term Memory....18

© BB Inventions Inc., 2022

Chained 18
Cross-Set 18
Challenge-Based 19
Inaction 19
Applying of Rules 19
Adding, Modifying, and Removing Rules 20
Adding a Rule 20
Modifying a Rule 21
Removing a Rule 22
Peterson 23
Generation 23
Tracking 24
Entity Selection 24
Obtaining Preselection Data 24
Relevant Score Distribution(s) 25
Peterson 25
Processing of Preselection Data 25
Performing The Selection 26
Roles 26
Granting Types 26
Automatic 26
Determined 26
Rotated 27
Staking 27
Economy-Based 28
Types 28
Baseline Role 28
Criteria & Restrictions 28
Requester & Partner 29
Criteria & Restrictions 29
Pre-Transaction Information 29
Transaction Fees 30
Feeless 30

© BB Inventions Inc., 2022

Adaptive 30
Static 30
Type-Based 30
Dynamic 31
Partner Selection 31
Providing 33
Generator & Enforcer 34
Criteria & Restrictions 34
Pre-Block Information 34
Enforcer Selection 35
Providing 38
Cross-Validating 38
Local Database(s) Updates 39
Dynamic Block Rewards 39
Speedster 41
Criteria & Restrictions 41
Thresholds 41
Calculation of New Threshold Value(s) 42
Threshold Implementation Types 42
Upper Threshold 43
Lower Threshold 43
Pre-Poll Information 43
Poll Creation & Voting 43
Sentinel 44
Criteria & Restrictions 45
Pre-Poll Information 45
Poll Creation & Voting 46
Special Types 47
Full, Lightweight & Storer Roles 47
Lightweight / Storer Entities Relationship 48
Lightweight vs Full vs Storer Entities: Block Rewards Differences 48
Blacklisted 49
Proof of Justice-Enabled Mechanisms 49

© BB Inventions Inc., 2022

Parameters...49
Dedicated Node...50
Specialized Subnetworks...51
Proof of Justice - Advantages...51

Introduction

Blockchains and similar peer-to-peer networks are systems comprising databases that are consensually shared across a plurality of entities and are actively maintained to track provided transactions. A decentralized blockchain enables one or more types of relevant activity and information to be shared between the plurality of entities. Thus, to ensure data legitimacy, said entities are in need to achieve a consensus over the authenticity of said shared relevant activity and information.

Prior Art & Issues

Proof of Work

One method for achieving a consensus well-known in the art is a concept called proof of work (PoW) and its derivatives, which are used in blockchains such as Bitcoin™ and the like. PoW generally relies on a first entity proving that a certain amount of computational effort was made to validate a plurality of transactions comprised in a PoW blockchain. One or more second entities are then tasked to validate if the effort produced by the first entity is sufficient, a result of said validations thereby enabling an eligibility for the first entity to receive a reward. To ensure the security of a system using PoW, complex cryptographic algorithms are often used, and the amount of computational effort needed for attaining a consensus between entities participating in the blockchain is often significant. From a security standpoint, PoW generally relies on entities, such as the first entity, validating transactions in the blockchain to invest a large quantity of computational resources in order to perform said validation, often leading to significant electricity costs. As a byproduct of the amount of said computational resources needed to perform said validation, a limiting participating barrier for less powerful entities often takes form. Thus, a system using PoW may not be appealing due to various reasons, such as an increasing requirement of powerful computer hardware in order for entities participating in the consensus to maintain a reasonable reward eligibility upon validation of transactions, as an increase in computational power to validate transactions is often seen in correlation with an increase of one or more types of blockchain activity, thereby often leading to both significant energy consumption and increasing monetary costs. Moreover, from a security standpoint, PoW generally relies on entities making transactions in the blockchain to invest resources, such as monetary resources, when making said transactions, which can lead to significant costs, oftentimes creating a limiting barrier in the type, nature, and scope of transactions said entities can make.

© BB Inventions Inc., 2022

Proof of Stake

Another well-known method for achieving a consensus is a concept called proof of stake (PoS) and its derivatives, which are used in blockchains such as Cardano™ and the like. Unlike PoW, PoS instead generally relies on selecting a validator to provide a block, thereby enabling an eligibility of a reward for doing so. The validator is typically selected proportionally to a quantity of invested resources, such as monetary resources. From a security standpoint, similarly to PoW, PoS generally relies on entities making transactions in the blockchain to invest resources when making said transactions, which can lead to significant costs, oftentimes creating a limiting barrier in the type, nature, and scope of transactions said entities can make. An advantage of PoS when compared to PoW is that it requires less energy to operate. PoS, however, tends to rely on entities investing a significant amount of one or more types of resource(s), often limiting benefits for less wealthy entities.

Proof of Space

Another well-known method for achieving a consensus is a concept called proof of space (PoSpace) and its derivatives, which is similar to PoW but tends to rely more on storage than heavy computational efforts, and which is used in blockchains such as Chia™ and the like. Proof of Space is a cryptographic technique where provers show that they allocate unused hard drive space for storage space. In order to work efficiently, Proof of Space is often used with Proof of Time. From a security standpoint, similarly to PoW and PoS, PoSpace generally relies on entities making transactions in the blockchain to invest resources when making said transactions, which can lead to significant costs, oftentimes creating a limiting barrier in the type, nature and scope of transactions said entities can make. Although, similarly to PoW requiring significant computational efforts, PoSpace requires the prover to have a significant amount of space and read speed in their data storing systems, such as a hard drive or a solid-state drive, thus often leading to a significant resources investment in relevant computer hardware.

There is therefore a need for a method and a system that will overcome the above-identified drawbacks.

Proof of Justice – Overview

Common problems in the technical field related to the current technology are that consensus algorithms often require a significant amount of one or more types of resource(s) to operate and that power in a related blockchain is often correlated to how wealthy a given entity is, often limiting decentralization by widening the gap between more wealthy entities and less wealthy entities. Proof of Justice provides, in one or more implementations, (i) an environmentally friendly, (ii) a fast, (iii) a decentralized, (iv) a feeless, and (v) a secured practical solution to these technical problems by combining multiple strategies, systems, and methods, including evaluating the reliability of entities and using trackable pseudo-random numbers.

© BB Inventions Inc., 2022

Base Architecture

Hooks

Hooks provide a way for executing rules and scripts while taking into account different states of the system. They are the foundation for how rules can execute and interact with Proof of Justice actions and events. Hooks essentially are a core defining factor about the order of rules execution within a given set of rules. Additionally, hooks can be chained together, so that one hook can invoke another hook.

Node

A node is a computing device that participates in the operation of a Proof of Justice-based system. This computing device is responsible for validating and committing transactions to the blockchain. In order to do this, the computer may either download the entire distributed ledger or delegate part of its storage under conditions. A given node has a unique private key / public key pair, i.e., a wallet, associated with it. A node is associated with an entity. For a blockchain implementing Proof of Justice, an entity will often be a public key / private key pair, i.e., a wallet. It should be noted that the term “node” and “entity” may be used interchangeably, and it is assumed the reader knows the difference between both and to which context each term applies. A node also has a mempool associated with it.

Mempool

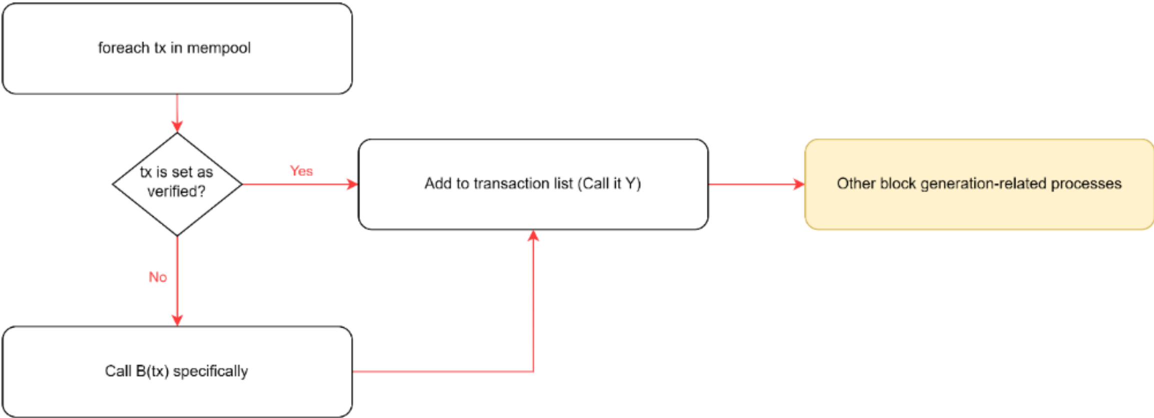
A mempool is a system containing to-be-confirmed transactions. These transactions have been recently submitted by nodes and have not yet been confirmed to be associated with a block. Hereinbelow is an example of a high-level overview architecture of a mempool.

© BB Inventions Inc., 2022

Receiving transaction in mempool

Background job, async (Call it B)

Transaction List generation for block generation-related actions



Transaction

A transaction is a set of data representing instructions coming from one or more nodes and targeting one or more other nodes, transactions, and/or other blockchain-related data. A transaction in Proof of Justice must be made in a specific manner in such that it respects a set of determined rules. Each transaction has at least one Partner Peterson and a nonce associated with it. Several types of transactions may be implemented, according to the Proof of Justice implementation. Different types of transactions can each have different sets of criteria to respect in order to be considered valid.

© BB Inventions Inc., 2022

Block

A block is a collection of transaction information data. Blocks are created by qualifying Generator entities, who add them to the blockchain by verifying new transactions and adding them in said blocks. A blockchain block has several sections.

Transactions

The first section contains all transactions and is further divided into several subsections, each subsection being associated with one or more transaction types, according to the implementation of Proof of Justice.

Peterson(s)

The second section contains the Peterson of the Generator associated with the block at the time of the block generation. Although uncommon, in some implementations of Proof of Justice, there can be several Generator Petersons per block.

Scoring System

The Proof of Justice (PoJ) consensus algorithm is a system that encourages entities to act justly by penalizing those that don't. This is done by giving each entity a score based on its behavior and then applying rules that limit the ability of malicious entities to repeatedly act in bad faith.

Score

Each entity in Proof of Justice is associated with a reliability, said reliability being established as a result of diverse actions and/or inactions. A score and a reliability of a given entity are influenced by one or more sets of rules. A given rule ϕ is associated with a corresponding weight γ, which can be provided in a form of a number, and with a given corresponding indicator λ, which can be either positive or negative. Below is listed an example set of rules, each rule ϕ being associated with a corresponding weight γ and a corresponding indicator λ. It is important to understand that Proof of Justice is not intended to be limited by the below-listed rules, weights, and indicators and that various other rules, weights, and indicators may be provided, possibly influencing a score and the reliability of entities part of the blockchain. As implementations of Proof of Justice vary greatly, the below-listed rules need to be customized on a per-implementation basis in order to achieve the full potential of Proof of Justice.

Table 1. Exemplary set of rules, each rule with its corresponding weight and indicator

Rule overview	Rule number ϕ	Weight γ	Indicator λ

© BB Inventions Inc., 2022

Having generated a different Peterson than an entity's Partner (as a Requester)	1	1	+
Having spent a determined time (and/or quantity of at least one generated block) in the blockchain since existence thereof.	2	2	+
Having generated a Peterson that is not frequent in a newly generated block	3	1	+
Generating a block	4	3	+
Acting as an Enforcer	5	3	+
Having generated a different Peterson than an entity's Requester (as the Partner)	6	1	+
Having generated the same Peterson as an entity's Partner (as a Requester)	7	1	-
Having generated the same Peterson as an entity's Requester (as a Partner)	8	2	-
Frequently using the same Partner	9	3	-
Being related to a Peterson which is frequent in a block's transactions	10	2	-
Having similar wallets in transactions in blocks (as a Generator)	11	1	-
Having an average score in transactions in a block under or above a determined value (as a Generator)	12	1	-
Including its own transaction in a block (as a Generator)	13	2	-
Having the same Enforcer within a determined number of previous blocks (as a Generator)	14	3	-
Having the same Generator within a determined number of previous blocks	15	3	-
Having an anomalous behavior as an Enforcer, a Generator, a Requester, and/or a Partner determined by machine learning and/or any other relevant identification algorithm	16	3	-

Role Granting

Referring to Figure 1, an exemplary score distribution of a plurality of entities is presented in a form of a normal distribution. Depending on the behavior of the plurality of entities and on the set of rules, the distribution may be provided in various forms. A score associated with a given entity, in one

© BB Inventions Inc., 2022

implementation of Proof of Justice, can be calculated using the following simple formula: $\sum \lambda\gamma\phi$, which translates to a sum of weights and indicators associated with a plurality of rules. Once the score is calculated for each entity, a reliability can be calculated by mapping the score to a probabilistic distribution. In one implementation of Proof of Justice, entities with a reliability below x1 are considered to be blacklisted. As such, according to Figure 1, all entities with a reliability below x1 will not be allowed or will be extremely limited, to perform any action in the blockchain until a rule, such as rule ϕ=2 in table 1, has passively increased the score above the blacklist threshold. This provides an extra layer of security as malicious entities will not be able to participate and/or profit from their malicious actions.

In Proof of Justice, different score thresholds define different levels of permissions and/or abilities for entities within the blockchain, i.e., roles. For example, a first threshold x1 may be used to blacklist entities, a second threshold x2 may be used to determine the Requesters, a third threshold x3 may be used to determine the Partners, a fourth threshold x4 may be used to determine the Generators and a fifth threshold x5 may be used to determine the Enforcers.

Some implementations of Proof of Justice may use rules enabling a given entity to only be allowed to act as one or more specific roles for up to a determined period of time, before losing access and/or having to complete a challenge in order to regain access. This allows for a more decentralized and secure system, as no entity can monopolize the actions within the blockchain.

When an entity completes an action that is not allowed by its current level of permissions, the entity may be penalized on top of seeing its request to perform such action denied. The penalties may take various forms, such as, for example, score reduction, blacklisting, removal of permissions and/or abilities, and/or any other suitable form of punishment.

Figure 1.

© BB Inventions Inc., 2022

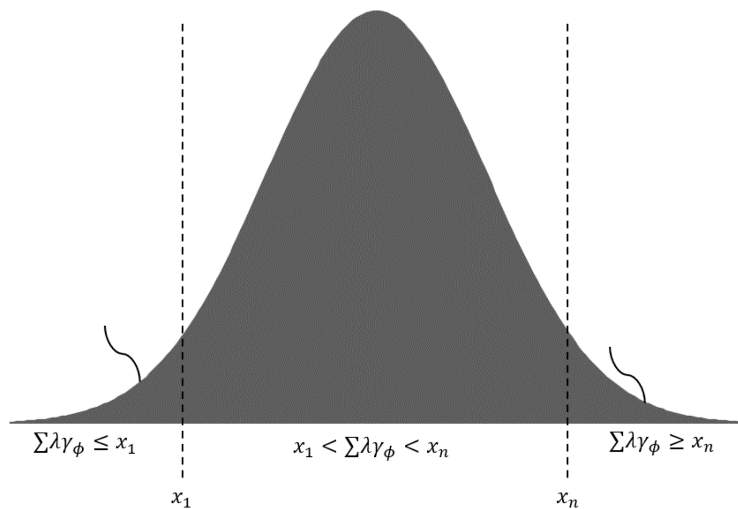


Figure 1

Levels

The score system of Proof of Justice is not bound to a single score per entity. Indeed, a plurality of scores can work together to achieve a more granular and accurate representation of a reliability of an entity. Much like a video game, the plurality of scores can be represented in a form of "XP" and "Level". The level is a way of representing the global, long-term performance of a given entity. The XP (Experience Points) is a representation of how close an entity is to reaching the next level, i.e., how well is the entity behaving recently.

© BB Inventions Inc., 2022

The levels and experience points are useful for two main reasons:

1. It makes it easier for entities, such as Enforcers and validators, to quickly identify which entities are trustworthy and which entities are not.
2. It allows the network to accurately represent the reliability of a given entity, compared to some other Proof of Justice implementations where the single score of an entity would be reduced cyclically instead to force a rotation of roles and prevent stagnation.

Once the "XP" of a given entity reaches the maximum allowed value, said entity gains a level, i.e., increment a second score by 1. The rate at which a given entity gains "XP" is determined by the parameters of the system. When an entity gains a level, its XP is reset to 0. The higher the level, the more difficult it becomes to gain XP, thus levels, as the rule score rewards slow down.

This system allows entities that have been reliable in the past to be forgiven for small, accidental mistakes. It also incentivizes entities to stay active in order to maintain their privileges. Indeed, such a system works well with rules penalizing entities that stay offline for long periods of time.

The levels and experience points can also be used in conjunction with other rules, such as Long-Term Memory and Challenge-Based rules, to create more complex policies.

Role-Dependent Levels

As it will be explained below, the reliability of a given entity, represented in the form of one or more scores, determines what actions said entity can and cannot do in the Proof of Justice-based system, said actions being determined by roles granted by the one or more scores thereof. This is especially important for security-related actions, such as enforcing or generating a new block. In order to ensure that the most reliable entities in the network are the ones performing those actions, a role-dependent level system can be used. In such a system, an entity's level would not only be determined by its global performance but also by its performance in the specific role said entity is fulfilling. For example, if an entity has a high level as an Enforcer, it would be more likely to be chosen as an Enforcer than another entity with the same global level but has a lower level as an Enforcer. This system ensures that only the most reliable entities are chosen for the most important roles and that those entities are the ones with the most to lose if they act maliciously. The role-dependent level system also incentivizes entities to participate in a variety of roles in order to have a higher chance of being chosen for the more important roles. Indeed, if an entity only ever participates as an Enforcer, it would have a lower chance of being chosen as a validator than another entity that participates both as an Enforcer and a Partner. This system encourages entities to be well-rounded and to participate in the network in a variety of ways, which is beneficial for the overall security and stability of the network.

Possible Drawbacks

One potential drawback of the levels and experience points system is that it may be difficult to fine-tune the parameters of the system in order to ensure that it is secure and effective. Indeed, if the parameters are not chosen correctly, it may be possible for entities with malicious intent to exploit the system and gain a high level without actually being trustworthy. Furthermore, if the parameters are not chosen

© BB Inventions Inc., 2022

correctly, it may be possible for entities that are trustworthy to be punished more severely than entities that are not.

Preventing Exploits

There are a few ways to prevent entities from exploiting the levels and experience points system.

One way to do so is to implement a Long-Term Memory rule which keeps track of the actions a given entity takes in the system and to use that information to re-evaluate one or more levels / XPs. For example, if an entity is chosen as an Enforcer but often fails to enforce blocks, its Enforcer level would be lowered.

Another way to do so is to change the role-dependent levels to be short-term only, instead of permanent. This way, an entity would have to continuously perform well in order to keep its high level.

A third way to do so is to use a Challenge-Based rule which would require entities to periodically prove their trustworthiness, such as with a challenge-based rule, in order to maintain their levels / XPs.

Resource Access Granting

One of the benefits of the score system in Proof of Justice-based systems is that it can be used to grant entities access to different resources and/or features based on criteria. For example, if an entity has a high level of enforcement, it may be granted access to specific parts of the blockchain, such as a faster DHT than the traditional Kademlia with other drawbacks, like Koorde, and/or even a subnetwork of only entities above a given score, i.e., Enforcers, making communications more efficient. This is beneficial because it allows entities that are more trustworthy and have proven their trustworthiness over time to gain multiple privileges, which incentivizes those entities to continue participating in the system.

Entity Reporting System

An entity reporting system is a system where entities can vote that other entities are malicious in order to reduce their score(s). This system incentivizes entities to report other entities which they believe to be malicious, as they will be rewarded if they voted correctly. This system is beneficial as it helps to reduce the overall level of trustworthiness of malicious entities and makes it more difficult for them to exploit the system. Furthermore, this system encourages entities to be vigilant and cooperate in order to ensure the security of the network.

In order to prevent a first entity from falsely reporting another entity in order to reduce its score and make it more difficult for it to participate in the network, one could require the first entity to provide proof of maliciousness when it reports another entity and punish it if, according to other entities in the network and/or a non-deterministic rule, it was wrong. This way, entities would need to have a good reason for reporting another entity, and they would be less likely to falsely report an entity.

© BB Inventions Inc., 2022

An additional way to do so would be to limit the number of reports an entity can make in a given period of time. This way, entities would need to be more selective in the reports they make, and they would be less likely to make false or frivolous reports.

One could also implement a fee-based approach in order to prevent spamming of the reporting system. This way, entities would need to pay a small fee in order to make a report, and this would discourage them from making false or frivolous reports.

One could also use a reputation-based approach in order to give more weight, such as rewards upon correct voting, to reports made by entities that have been proven as trustworthy / have a good report reputation and less weight to reports made by entities that have been proven as untrustworthy / to have a bad report reputation. This way, it would be more likely for legitimate entities to have their reports taken seriously, while entities which have a history of making inaccurate reports would be less likely to have their reports taken seriously.

Finally, one could use a collateral approach in order to make it more costly for an entity to falsely report another entity. This way, the entity would need to put up some sort of collateral, such as native cryptocurrency, in order to make a report. If the report is found to be false, then the entity would lose its deposit.

Types of Rules

Rules are scripts that take one or more given inputs and return an adjusted score for one or more entities. Rules can take various forms and are of various types.

Regular

Regular rules are rules which listen to a hook and when an event is fired, calculate the adjusted score of one or more entities based on said event. A regular rule can be seen as a kind of if-then statement. If a certain event is fired, then the score of an entity is increased or decreased by a certain amount according to an evaluation of said event. For example, a regular rule could be written such that if an entity generates a new (valid) block, then its score is increased by 3 points. Another example of a regular rule could be one written such that if an entity tries to generate a block that is considered malicious according to a given set of rules, then its score is decreased by 10 points. There are an infinite number of ways in which regular rules can be written, making them very versatile.

Non-deterministic

Non-deterministic rules in Proof of Justice are rules which listen to one or more hooks and when an event is fired, attempt to predict the adjusted score of one or more entities based on said event and, in some cases, previous events. A non-deterministic rule is a special type of rule which is allowed to force an adjustment of score for a given entity with a delay, although may be subjected to the supervision of other entities, such as entities with the Sentinel role. Non-deterministic rules are subject to changes and

© BB Inventions Inc., 2022

adaptations and are generally much more space and computationally resource-consuming than other types of rules.

Non-deterministic rules are used to predict if the behavior of an entity is malicious, and to suggest possible outcomes of events, such as an adjusted score, a blacklist, or any other relevant restriction(s). They can be used to make recommendations or to take actions on behalf of an entity. Non-deterministic rules are often used in situations where it is difficult or impossible to determine all of the relevant information about an event or situation. Non-deterministic rules can also be particularly useful in opinionated rules.

The dataset of a non-deterministic algorithm should be publicly available and historical data on the Proof of Justice-based system; in the case of a Proof of Justice-based blockchain, transactions in blocks. As such, generating an algorithm capable of reliably identifying malicious actions by entities in the network can be a troublesome task due to the lack of an initial dataset. Thus, it is crucial to keep in mind that as much network-related and blockchain-related information has to be recorded as possible in order to have a reliable non-deterministic algorithm. It is also strongly recommended to create a working prototype able to run simulations of close-to-real-world entities' behaviors, including as many known malicious behaviors and actions as possible. It would then be possible to generate a dataset using said simulations and thereby train a non-deterministic algorithm.

It is also recommended to include, at release, a sort of "Trial" phase for a Proof of Justice-based system implementing a non-deterministic algorithm of the like, wherein said "Trial" phrase would allow one or more system managers to reverse potentially breaking decisions made by a non-deterministic algorithm while allowing the collection of real-world data in order to increase the performance of the non-deterministic algorithm.

It is also recommended to prioritize both efficiency and precision of the algorithm. Indeed, making an algorithm that is precise but computationally intensive to run would both lead to centralization of the network towards stronger computing nodes, but also defeat the philosophy of a low carbon footprint consensus algorithm.

Requirement

Requirement rules are rules which, for a given Proof of Justice-related action, are required to execute successfully and return a positive value before one or more entities can execute said Proof of Justice-related action.

For example, a requirement rule for the action of generating a block could be that the Generator would first need to either:

1. Possess a certain amount of the native cryptocurrency; or
2. Have been active in the network for over a determined period of time.

Requirement rules can help prevent several types of attacks. By mandating that new entities possess a certain amount of cryptocurrency before being able to perform specific actions in a Proof of Justice system, it is more difficult and expensive for an attacker to create many new identities, and, in this

© BB Inventions Inc., 2022

example, also reward long term users by up to eliminating such possession requirements. As such, requirement rules add an important layer of security to Proof of Justice systems.

A great example of a requirement rule (and challenge-based rule, as it will be elaborated upon in the "Scoring System -> Types of Rules -> Challenge-Based" section) which would greatly improve a Proof of Justice-based system in a decentralized manner would be a rule aiming at authorizing a given entity to gain access to one or more portions of the Proof of Justice-based system, wherein said requirement rule would comprise:

1. obtaining a set of instructions to be executed by the given entity, the set of instructions being suitable for authorizing the given entity to gain said access;

2. a portion of the entities in the network approving the obtained set of instructions;

3. providing the approved and obtained set of instructions to the given entity;

4. obtaining one or more results from an execution of the provided set of instructions by the given entity; and

5. granting the access in the network to the given entity using said one or more results.

It will be understood that the execution of the set of instructions involves receiving an input from the user being the owner of the given entity. In one or more implementations, the set of instructions is suitable for ensuring that the user is not the owner of an existing entity in the network. In one or more associated implementations, it should be noted that a given entity can only be associated with one, or a determined number thereof, computing node.

In one or more implementations, a requirement rule may not affect the score(s) of a given entity, but enables an access to one or more elements of the Proof of Justice-based system, such as sending transactions, accessing a specialized DHT algorithm, and/or the like. Thus, the requirement rule could enable an execution of one or more actions of one or more roles. In one or more implementations, complying or failing to comply with the requirement rule may result in an entity having to provide data enabled by the processing of one or more defined instructions.

It will be understood that one or more instructions are related to the requirement rule, i.e., that the score(s) of one or more entities may be affected by the data enabled by the processing of said one or more instructions.

Score-Dependent

Score-Dependent rules are rules which can be of any other type but are only enabled when the reliability score of a given entity is superior to a given threshold. For example, a score-dependent rule could be "If X has a score above Y, then do Z". In this way, different conditions can be put in place depending on the score of a given entity. This allows for a more flexible system where different

© BB Inventions Inc., 2022

conditions can be applied in different situations. A score-dependent rule is used to more granularly evaluate entities.

For example, a Score-Dependent rule could reduce the score of an entity every n^{th} time said entity acts as an Enforcer, forcing a rotation of entities and preventing stagnation.

Adaptive

Adaptive rules are rules which can be of any other type and which can adapt themselves automatically according to an evaluating of one or more aspects of the blockchain. For example, a given adaptive rule may adjust its own weight based on specific type(s) of network activity or any other relevant trigger. The purpose of such rules is to make the Proof of Justice blockchain more resilient against malicious actions exploiting a rule weight/behavior itself. Additionally, adaptive rules may help the network to automatically correct certain issues without the need for manual intervention.

Action-Dependent

Action-Dependent rules are rules which can be of any other type but are only enabled when a given entity is performing one or more specific Proof of Justice-related actions. For example, an action-dependent rule could be "If X is trying to do Y, then Z must happen first". In this way, different conditions can be put in place depending on the actions being taken by a given entity. This allows for a more flexible system where different conditions can be applied in different situations. An action-dependent rule can be used to prevent certain actions from being performed unless certain other conditions are met first. For example, if an entity has generated a block, an action-dependent rule could be used to ensure that the Generator has selected the Enforcers in a correct manner and if not, punish and reduce its score.

Long-Term Memory

Long-Term memory rules are rules which listen to one or more hooks and when an event is fired, according to a given number (or all) of previous events of one or more types for one or more given entities part of the blockchain, calculate the adjusted score of one or more entities. The Long-Term memory rules are important because they help keep the blockchain running smoothly by adjusting entities' scores according to their long-term behavior.

For example, let's say we have a long-term memory rule that is listening to the "enforcement_done" hook. This rule could be fired every time an enforcement is completed when a given entity is generating a given block. The rule would then take into account all of the entity's previous enforcements and, for example, if the entity is often selecting the same Enforcers or Enforcers with similar characteristics (for example similar machine specs), then said entity may be punished and its block may be deemed invalid. These events can also be used by other rules in the blockchain to determine how that entity should be treated going forward.

© BB Inventions Inc., 2022

The Long-Term memory rules are important because they provide a mechanism for the blockchain to identify long-term patterns of behavior and react accordingly. This can help the blockchain avoid situations where entities are able to game the system or otherwise exploit long-term loopholes.

Short-Term Memory

Short-Term memory rules are rules which are very similar to Long Term Memory rules but work with a more limited set of data. Specifically, Short-Term memory rules only take into account the immediate past when making decisions. This is in contrast to Long Term Memory rules which can take into account historical data.

The advantage of Short-Term memory rules is that they can be evaluated much faster than Long-Term Memory rules, and can also identify short-term patterns that may not be efficient or even possible, according to their implementations, for Long-Term Memory rules. For example, if an entity has a very good history of just behavior, but recently started sending large chunks of funds to several other wallets and acts maliciously when selecting Enforcers upon generation of a block, said entity could be as much as blacklisted directly, as it would probably indicate a hack of said wallet, potentially giving time to the owner of the wallet to take action against the malicious user outside of the blockchain.

Chained

Chained rules are rules which listen to one or more rules outputs (and, in some cases, one or more hooks) and when an event is fired, calculate the adjusted score of one or more entities based on said event.

One common use case for chained rules is to give entities a score penalty if they have been flagged as malicious by one or more rules, such as non-deterministic rules. This can be accomplished by setting up a rule which triggers on the output of other rules, and gives the entity a score penalty if it meets certain criteria. For example, this could be accomplished by having a rule which triggers on the outputs of two or more other rules, and gives the entity a score penalty if both those rules have fired within a certain time period.

Cross-Set

Cross-Set rules are rules which listen to one or more hooks and refer to one or more rules in a different set of rules and when an event is fired, calculate the adjusted score of one or more entities based on said event.

For example, you have a set of rules A applying only when a given entity is acting as an Enforcer, and you want to take the score from set A and input it to a Cross-Set rule. The Cross-Set rule would listen for an event in Set A and upon firing of said event, calculate the adjusted score for that entity.

Cross-set rules are essentially used when you want to maintain two or more independent sets of rules but still want to be able to share data between them.

© BB Inventions Inc., 2022

Challenge-Based

Challenge-based rules are rules which listen to one or more hooks and when an event is fired, require a subset of one or more entities to complete one or more challenges and based on the results, calculate the adjusted score of said one or more entities. A challenge-based rule is often used in conjunction with other rules, such as Long-Term Memory and non-deterministic rules.

For example, if a given non-deterministic rule predicts that a given entity is acting maliciously, a challenge-based rule could take said prediction as an input as well as any other relevant input, and require said entity to complete a challenge. If said entity does not complete the challenge successfully, then the challenge-based rule could adjust the score of said entity downward.

A challenge-based rule can be used to implement a wide variety of policies, such as rate limiting, pattern breaking, and so on. Additionally, challenge-based rules can be chained together to create more complex policies. For example, a series of challenge-based rules could be used to require an entity to complete progressively more difficult challenges in order to continue accessing a given resource if said entity is requesting several accesses over a determined period of time.

Inaction

Inaction rules are rules which will calculate the adjusted score of one or more entities if one or more hooks are not triggered under specific criteria, such as a period of time.

It is useful to think of inaction rules as a way of expressing "if X does not happen, then do Y". This can be used to incentivize entities to stay active in the network and potentially filter out anomalous entities.

For example, let's say there is an inaction rule which states that if an entity does not participate as a Partner or Enforcer at least once over a determined period of time, its score would be decreased. If an entity stops participating for an extended period of time, it would see its score significantly decrease and would have to work all over again to obtain privileges, such as roles, in the network.

Applying of Rules

Once a given entity receives a new block from the network and finishes validating said block, it will execute rules of all sets in the determined order, said order being determined by one or more criteria, such as one or more Proof of Justice parameters, the hooks execution order and/or the like. In most Proof of Justice implementations, rules have a number associated with them, which classifies the order in which they should be executed. This also goes for the set of rules itself, in a case of a plurality of sets of rules in the same Proof of Justice-based system. Such an implementation allows to always be able to reproduce and reconstruct the score(s) associated with every entity part of the network at any given block state for a given chain version.

© BB Inventions Inc., 2022

Adding, Modifying, and Removing Rules

Dynamic adding, updating, and removal of rules allows a Proof of Justice-based system to be constantly improved and refined, ensuring that it remains fair and effective. Additionally, this flexibility allows to respond quickly to changes, attacks, and events, making the system more resilient and secure.

Adding a Rule

When entities want to add a new rule to the blockchain, they can create a poll where other entities with a score above a first threshold can vote whether they accept to add said rule. If the poll reaches the necessary quorum, then the new rule is added to the blockchain. Adding a new rule in this manner allows for a decentralized and democratic way of governing the blockchain, while still allowing for a certain degree of flexibility and ensuring that votes are made by legitimate entities.

The process of adding a new rule to the blockchain via poll can be as follows:

1. An entity creates a poll with the proposed new rule and submits it to the network.

2. Other entities with scores above a second threshold (and/or determined as suitable rule code reviewers by other voting processes), which may be the same as the first threshold in some implementations, review the proposed new rule and cast their votes.

3. If the poll reaches a quorum, then the new rule is added to the blockchain. A quorum can be reached if a certain percentage of entities vote in favor of the new rule (e.g., 51% or two-thirds).

4. The new rule is now added to the rule library, relevant hooks are now in place and the rule is now in function.

Various techniques and strategies can be implemented in order to prevent possible exploits of such a system, such as:

1. Implementing a fee to create such a poll

2. Penalizing entities in the minority at the end of the voting period / upon reaching quorum (e.g. through a slashed score)

3. Weighted voting, where entities with higher scores have more weight in the voting process

4. Allowing for a certain period of time after the poll is created for entities to review the proposed rule before voting.

5. Enabling a reporting mechanism where entities can provide valid proof(s) that said rule is malicious and get significantly rewarded if they were right (and punished if they were wrong, avoiding bloating), and strongly penalizing the entity which created said poll. Optionally, such a

© BB Inventions Inc., 2022

reporting mechanism could also include a "testing phase" for newly created rules, where the rule is enabled and active but its effects are (partly) reversible for a given period of time.

6. Only allowing specific entities to start a rule creation poll in case of a semi-decentralized system

7. Only allowing entities successfully resolving one or more challenges emitted by the network / one or more entities to create a rule creation poll

Here is an example of a working system for adding new rules (where a mechanism similar to the abovementioned reporting mechanism is enabled for entities identifying malicious binaries and source code):

1. The “Rule Proposal Accepted” flag not being raised, an authorized entity, such as a system manager or an elected entity, decides that it is necessary and/or helpful to introduce a new rule.

2. Said authorized entity opens a rule creation poll, where other qualifying entities part of the blockchain can vote whether they would like a rule that does what the new rule says it would do (a summary of the rule itself, in words).

3. If a positive quorum is reached, the “Rule Proposal Accepted” flag is raised, indicating that the network accepted to implement such a rule.

4. When the rule script has been developed and if the “Rule Proposal Accepted” flag is raised, a “Rule Review” poll is created and contains:
    a. The rule script source code;
    b. The binary files for different platforms;
    c. The hashes of the script source code and binaries files; and
    d. A pointer to the poll which raised the “Rule Proposal Accepted” flag.

5. If a positive quorum is reached, the appropriate binary files are placed in the appropriate rules set, the appropriate participating entities are rewarded, and the “Rule Proposal Accepted” flag is reset.

Modifying a Rule

If it is decided that a rule needs to be changed, a “Rule Modification Proposal” poll can be created. The process for modifying a rule would be as follows:

1. The “Rule Modification Proposal” poll is created and contains:
    a. The proposed new rule script source code;
    b. The proposed new binary files for different platforms;
    c. The hashes of the proposed new script source code and binaries files; and

© BB Inventions Inc., 2022

    d. A pointer to the poll which raised the “Rule Proposal Accepted” flag for the original version of the rule being modified.

2. If a positive quorum is reached, the “Rule Proposal Accepted” flag is raised, indicating that the network accepted to implement such a rule change.

3. When the new rule script has been developed and if the “Rule Proposal Accepted” flag is raised, a “Rule Review” poll is created and contains:
    a. The new rule script source code;
    b. The new binary files for different platforms;
    c. The hashes of the new script source code and binaries files; and
    d. A pointer to the poll which raised the “Rule Proposal Accepted” flag.

4. If a positive quorum is reached, the appropriate binary files are placed in the appropriate rules set, the appropriate participating entities are rewarded, and the “Rule Proposal Accepted” flag is reset.

Removing a Rule

If it is decided that a rule needs to be removed, a “Rule Removal Proposal” poll can be created. The process for removing a rule would be as follows:

1. The “Rule Removal Proposal” poll is created and contains:
    a. The rationale for why the rule should be removed; and
    b. A pointer to the poll which raised the “Rule Proposal Accepted” flag for the original version of the rule being removed.

2. If a positive quorum is reached, the “Rule Proposal Accepted” flag is raised, indicating that the network accepted to remove such a rule.

3. When the new rule script has been developed and if the “Rule Proposal Accepted” flag is raised, a “Rule Review” poll is created and contains:
    a. The new rule script source code;
    b. The new binary files for different platforms;
    c. The hashes of the new script source code and binaries files; and
    d. A pointer to the poll which raised the “Rule Proposal Accepted” flag.

4. If a positive quorum is reached, the appropriate binary files are placed in the appropriate rules set, the appropriate participating entities are rewarded, and the “Rule Proposal Accepted” flag is reset.

5. If a negative quorum is reached, the “Rule Proposal Accepted” flag is reset.

© BB Inventions Inc., 2022

Peterson

The core of Proof of Justice's operations and actions is based on pseudo-random, trackable numbers named Petersons. A Peterson is, in most implementations, the result of a combination of a block's Peterson and an entity's identifier.

Generation

A Peterson (P1) is obtained by combining the (or one of, within a determined range) previous block's Peterson (P2) with the identifier of the entity generating the new Peterson. In one or more implementations of Proof of Justice, the process would go as follows:

1. The entity obtains P2;
2. The entity obtains its ID;
3. The entity split its ID into equal parts matching P2's length;
4. The entity maps each part of its ID to numerical values; and
5. The entity XORs P1 with the 1st part of the ID, then XORs the result with the second part, and so on up until the last part of the ID has been XORed with the previous XOR result.

In some situations, a 6th step where the entity signs P1 is required. There could also be a need for multiple entities to sign it, depending on the implementation and situation.

It will be understood that the steps can vary greatly based on the implementation. For example, the entity may not need to split its ID into several parts. It could be that the entity only needs to XOR its ID with P2 once, or even use an entirely different mechanism without XORing altogether. In some implementations, one may have the ID provided and/or generated by one or more other entities in the network.

In some implementations, instead of using the ID of the entity directly, a dynamic generation based on the entity's score and its own ID could be used thereof.

It is also possible for a block to have a plurality of Petersons associated thereof. For example, the block's Generator could generate n Petersons, where n is the nth last block at the moment of generating said block. Then, an additional numerical reduction step would be done in the obtaining process in order to obtain P2 from said plurality of Petersons.

Another implementation could require, in a block generation process or a transaction generation process, a block's Generator/transaction's Requester to obtain one or more Petersons from one or more other entities, and then include those in said block/transaction.

It would also be possible, for a given implementation, to dynamically change the Peterson's generation algorithm according to one or more criteria, such as a score of the entity, a determined number of blocks that have been generated since existence of the blockchain, and/or the like.

© BB Inventions Inc., 2022

Tracking

The purpose of a Peterson is to be able to identify when its generation was made, i.e., what was P2 at that time, in order to have a kind of decentralized time measure to rely on and to establish truth and trust in the system. Indeed, this is why most implementations of Proof of Justice will choose a solution like the abovementioned XOR's one in order to recover P2 using a known variable, such as the ID in said abovementioned solution.

An entity can verify the generation time of a Peterson by checking if P2 is what it should be at that time, according to its knowledge of the network. For said abovementioned solution, the tracking would go as follow:

1. The entity obtains P1;
2. The entity obtains the ID which was used in P1;
3. The entity splits the ID into equal parts matching a Peterson's length;
4. The entity maps each part of its ID to numerical values; and
5. The entity XORs P1 with the last part of the ID, then XORs the result with the second last part, and so on up until the first part of the ID has been XORed with the previous XOR result, to obtain P2.

If P2 is what it should be at that time, according to the situation for which the tracking was done in the first place, then the generation of the Peterson can be considered valid. If not, the generation of said Peterson can be considered invalid.

It will be understood that the tracking steps can vary greatly based on the generation's implementation.

Entity Selection

Most processes and actions in a Proof of Justice-based system require selecting a specific range of entities in order to communicate/interact with one or more of them. This is done as a security measure, limiting the possibility of a given entity to always communicate with one or more other specific entities for one or more actions of one or more roles as an attempt to perform potentially malicious deeds. Entity selection is often performed for selecting one or more entities with a right to a given role (R1).

Obtaining Preselection Data

The first step in performing an entity selection is to obtain the preselection data. Preselection data is composed of two things:

1. The current, relevant score distribution(s) (S1) to select the target; and
2. The current Peterson (P1) of the entity in need of performing the selection.

© BB Inventions Inc., 2022

Relevant Score Distribution(s)

S1 can vary greatly according to the implementation of Proof of Justice. In one implementation using Levels and XP, S1 may be the Levels of the entities for R1.

In another implementation that is using a single score per entity, then the whole score distribution would be obtained and declared as S1. In this case, a subset (S2) of S1 would be extracted, where S2 contains all entities which can be categorized as R1.

In another implementation that is using a plurality of scores per entity, each score being associated with a corresponding set of rules, then a weighted approach of one or more of those subsets, according to the nature and purpose of the selection, could be compiled, and thereby form S1.

It will be understood that further filtering on S1 or S2 can be performed, according to the implementation of Proof of Justice. For example, an implementation could further select the top N entities in S2, N being a Proof of Justice parameter; S2 would then be used instead of S1.

Peterson

P1 is obtained using the abovementioned Peterson -> Generation process. P1 will be used as a randomization seed, allowing the tracking and validating of the entity selection process.

Processing of Preselection Data

Once S1 and P1 have been obtained, S1 is separated into several chunks, according to the number of entities comprised in S1. The number (N1) and size (Si1) of chunks can be determined in multiple different ways and according to multiple criteria, according to the implementation of Proof of Justice. In one implementation, N1 could be a Proof of Justice parameter while Si1 would be calculated accordingly. In another implementation, the opposite would apply; Si1 would be a Proof of Justice parameter, while N1 would be calculated accordingly. Small adjustments can be made to avoid empty chunk(s), based on the implementation, such that no chunk can have a Si1 smaller than X.

The chunks are then shuffled using P1 as a randomization seed. Then, the first n chunks are selected. N can be determined in numerous ways. For example, in one implementation could be a Proof of Justice parameter, while in another implementation it would be calculated according to the reason for the selection.

After the chunks have been shuffled and the first n selected, the entities of the selected chunks are gathered to form a set (E1). Depending on the implementation, E1 may or may not be organized in a trackable manner, although, when possible, such organization is recommended to enable further reproducibility. For example, in one implementation, E1 could be shuffled using P1 as a randomization seed. In such an implementation, it is further recommended to keep track of active and inactive entities in the network in order for such reproducibility to be relevant and usable. Otherwise, it would be inaccurate to assume that the entity performing the selection is truly selecting an entity in a sequential order, from the first one to the next, as it will be described below.

© BB Inventions Inc., 2022

Performing The Selection

Finally, an entity e is selected from E1 to perform R1. The selection of e is done by the entity in need of performing the selection. The process for selecting e could be as simple as randomly selecting one entity from E1 using P1 as a randomization seed. However, it could also be more elaborate, according to the implementation of Proof of Justice; for example, if multiple entities are needed to fill R1 (e.g., 3 validators), then e could be selected sequentially from E1 using P1 as a randomization seed until 3 distinct entities have been selected.

Roles

Proof of Justice's score is tied to the behavior and trustworthiness of a given entity part of the network. Achieving a certain level of trustworthiness allows a given entity to access new privileges and responsibilities, allowing it to be compensated with attractive rewards in exchange for its services. Roles can vary greatly from one implementation of Proof of Justice to another.

Granting Types

Automatic

One or more criteria may be used to define which entities are in the top tier, thus having access to a given role. For example, a criterion could be the entity being in the top X% of all entities with the best score(s) (such as the role's Level of each entity). Another criterion could be that if a score of a given entity is above a determined threshold, then the role is granted. For example, if an entity has a global reliability score above a threshold equal to 50, then it would unlock the right to act as, let's say, a Generator.

This allows avoiding that entities with poor scores (and, consequently, less likely to be worthy) have access to said roles while also incentivizing entities to continuously perform, "Grind", a given role to access more rewards (elaborated upon in the Roles -> Types -> Generator section).

A given role may have different requirements for each implementation. For example, a Generator role may need to be in the top 30% of all entities with the best score(s) (such as the role's Level of each entity) while a Partner role may only need to be in the top 70%, according to a given implementation.

Determined

In order to become a role in a given implementation of Proof of Justice, an entity must be approved by one or more other entities in order to access a given role. The approving entity(ies) will assess whether the entity has the necessary skills or qualities to be relevant for that position. For example, a "Code Reviewer" role may need to obtain the approval of X other entities in the network in the form of a voting, or simply by one or more system managers, according to the implementation. There may also be

© BB Inventions Inc., 2022

specific criteria that the entity must meet in order to be approved, such as proving to have relevant qualifications for that role by, for example for the "Code Reviewer" role, solving a programming challenge generated and proposed by said approving entity(ies) in a decentralized manner (wherein, in the voting example, said solution to said challenge and said challenge itself signed by one or more approving entity(ies) would be provided to the network at the moment of requesting the voting).

Rotated

In order to ensure that a given role is not monopolized by a handful of entities, said role may be rotated (meaning a subset of entities would gain access to said role, and another subset of entities would lose access thereof) periodically according to one or more criteria, such as:

1. Every X number of blocks, rotate Y% of all entities with a given role;
2. Every block with a pair Block Peterson, rotate Y% of all entities with a given role; and/or

any other relevant criterion.

By doing so, it is ensured that different entities have the opportunity to fill said roles and avoid stagnation and centralization of power within the network.

Staking

In order to ensure that a given entity is invested in the success of a given role's action, according to the implementation and the nature of said role's action, one or more assets of said entity would be frozen for a given amount of time, according to one or more criteria. This means that the entity will not be able to access or use said assets during said time period, which can be determined as a time distance between blocks (See Peterson -> Tracking). The nature and/or scope of the freezing may be determined according to various conditions, such as a given type and/or nature of said role's action, a quantity and/or nature of the one or more assets, a score of said given entity and/or a receiving/concerned entity of said role's action and/or the like.

As such, it is ensured that entities are invested in the success of role(s)'s action(s) and are less likely to act maliciously. As such, in one or more implementations, an entity determined to have acted maliciously would see:

1. One or more of its assets frozen for an extended period of time;
2. Its score(s), such as its level related to said action, be negatively impacted;
3. One or more of its assets distributed amongst a subset of entities in the network;
4. One or more of its assets "burned", increasing the block rewards therein in the case of a Proof of Justice blockchain; and/or

any other relevant penalizing thereof.

© BB Inventions Inc., 2022

Economy-Based

In some implementations of Proof of Justice, entities that want to take part in the network and/or perform certain actions related to it have to trade one or more assets with the system itself in order to gain a right to do so. The assets can vary greatly from one implementation to another. For example, in one, the assets could be in the form of the score(s) of an entity, while in another it could be in the form of tokens / coins. The trade is done in order to get access to a role and/or the right to perform a certain type of action related to the blockchain. For instance, an entity could trade a determined amount of assets, based on the supply and demands of other entities in the network, for getting access to said role's action or the role itself. In another example, instead of trading assets, one or more entities trade their own score instead. For instance, an entity with a global score of 100 could trade 40% of it to get a chance to act as an Enforcer role for a determined number of blocks.

This system ensures that entities are invested in the success of the network and are less likely to act maliciously. As such, in one or more implementations, an entity determined to have acted maliciously would see:

1. Its score(s) be negatively impacted;
2. One or more of its assets distributed amongst a subset of entities in the network;
3. One or more of its assets "burned", increasing the block rewards therein in a case of a Proof of Justice blockchain; and/or
4. Any other relevant penalizing thereof.

Thus, a system, in one or more implementations, using Proof of Justice, may help to create a more secure and trustworthy network by simulating a decentralized, economic-based system.

Types

Baseline Role

Every role implements and is derived from the Baseline superclass. This class provides the core nature of each role in a Proof of Justice-based system.

Criteria & Restrictions

One or more criteria must be met in order for a given entity to initiate a role-related action validly. These criteria can be of various natures, with the primary one being the entity having a valid score (such as its role's Level in a case where the Levels and XP system is used).

In one implementation, a challenge could have to be completed by the entity before being able to perform said role-related action.

Other techniques can also be used, such as implementing a rule where only a subset of entities in the network are allowed to perform said role-related action for a given range of blocks, said subset being

© BB Inventions Inc., 2022

dynamically calculated based on the Peterson of the current block. Although limiting, this technique could further enhance the security of a Proof of Justice-based system.

Restrictions can also be put in place for a given entity according to one or more criteria, varying greatly according to the concerned role. For example, a maximum quantity of role-related actions and/or a limiting of one or more aspects of said role-related action could be enforced based on the entity's role Level. As such, a lower Level would mean increased restrictions compared to an entity with a higher Level.

In a case where a plurality of entities of said role is required to cooperate in order to render a role-related action valid, then a combination of restrictions such as the abovementioned ones would, in an implementation, be used thereof.

Requester & Partner

The Requester role is responsible for generating and broadcasting transactions to the network. This role essentially allows a given entity to create and push a transaction, according to multiple criteria. A Partner role is responsible for validating parts of a transaction and providing its Peterson in order to render said transaction valid later on.

Criteria & Restrictions

Alongside the default criteria, one additional criterion appropriate for this role could be that the entity must not have done more than X transactions of one or more specific types in the last Y blocks/timeframe, wherein the timeframe can be calculated using the Peterson associated with each transaction (which will be elaborated upon below). In one implementation, X could be a Proof of Justice parameter, while in another it could be calculated using the average of transactions of those specific types done by other entities in the last Z blocks, where Z can be either Y, a Proof of Justice parameter or calculated dynamically using one or more rules.

Examples of aspects where restrictions would be appropriate to impose for the Requester role could include the number of assets the Requester can transfer according to its Requester Level, a sum of transferred assets over time (established using a score, said score being determined using a Long-Term Memory rule, for example), and/or the like.

Pre-Transaction Information

Pre-transaction information is data that is required to include in a transaction in order for said transaction to be valid. This data is generally generated and obtained before doing other transaction generation processes. Pre-transaction information can comprise things like the entity ID of the Requester (such as a wallet public key, if applicable in the implementation), the assets to transfer, the ID of the destination entity receiving the assets (such as a wallet public key, if applicable in the implementation), a nonce, the ID of the type of transaction and/or the like. Pre-transaction information can vary greatly based on the implementation of Proof of Justice.

© BB Inventions Inc., 2022

In several implementations, once the pre-transaction information has been gathered, it is then compiled and signed (PT1). In several implementations, the Requester also generates its Peterson (P1) and includes it in PT1 before signing PT1.

Transaction Fees

Feeless

A transaction made in a Proof of Justice-based system has the potential to be completely feeless mainly due to the Peterson mechanism, as it will be elaborated upon in the Generator role, and the reliable trustworthiness evaluation and establishing of entities within the system over time. In one implementation of Proof of Justice, all transactions would be feeless.

Adaptive

In another implementation, the fee associated with a transaction is adaptive, meaning it is feeless only according to specific criteria, such as:

1. The score(s) of the Requester, such as its Requester Level;
2. A challenge performed by the Requester as part of the pre-transaction information gathering process;
3. According to the token(s) type(s) involved in the transaction (for example, one could say that the native coin is feeless, while tokens may have access to feeless transactions based on one or more criteria, such as with an approval by a system manager, allowing the corporation which made the blockchain to potentially access additional revenue streams); and/or

any other criterion which may be deemed appropriate by the creator(s) of a Proof of Justice-based system.

Static

In another implementation still, the fee is static and unchanging regardless of any criteria. The fee could be, for example, a very low amount determined as a Proof of Justice parameter.

Type-Based

In another implementation, fees are type-based. This means that there would be a different fee for each type of transaction. For example, in a system where tokens can be sent without fees, one type of transaction could have a fee associated with it while another type of transaction (the sending of the native coin for example) would not have any fee. This implementation could be used in conjunction with any of the other implementations. For example, a system could have feeless transactions for its native coin but not for tokens, or it could have feeless transactions only when a challenge is completed successfully.

© BB Inventions Inc., 2022

Dynamic

Finally, in another implementation, fees are dynamic and calculated based on the fees of the last X blocks for either all transactions or a subset of transactions determined using one or more criteria (such as the type of transaction, the score(s) of the entities involved in the transaction, a challenge completed successfully by the Requester, and/or any other criterion which may be deemed appropriate). This implementation could also be used in conjunction with any of the other implementations. For example, a system could have feeless transactions for its native coin but not for tokens, or it could have feeless transactions only when a challenge is completed successfully.

It should be noted that these are only examples of how transaction fees can be handled in a Proof of Justice-based system and that any other method which may be deemed appropriate could also be used.

Partner Selection

In order for a transaction to be considered valid in a Proof of Justice-based system, a Peterson obtained from an entity with a role Partner (Pa1) must be obtained and comprised in said transaction.

According to step 202 in Figure 2, firstly, Pa1 is selected using the abovementioned Entity Selection process. The Peterson (P1) of the Requester, according to the implementation of Proof of Justice, is also fed to said Entity Selection process.

According to step 204, once Pa1 has been selected, the Requester sends a message (M1) to Pa1, wherein M1 contains PT1. Once Pa1 receives M1, it generates its own Peterson (P2) using the abovementioned Peterson -> Generation process. Pa1 then evaluates M1 and determines M1's validity by analyzing it thereof. The analyzing may vary greatly based on the pre-transaction information included therein. In most implementations, one important part of the validation is to evaluate the distance between P1 and P2. To do so, Pa1 would use the abovementioned Peterson -> Tracking process for P1, then determine the distance in blocks between P1 and the block Peterson used in generating P2.

According to step 206, Pa1 then includes P2 in a response message (M2). In a given implementation of Proof of Justice, M2 would also comprise additional information, such as the hash of at least one relevant part of M1. M2 is then signed and provided back to the Requester.

According to step 208, it is determined if the Requester is in need of one or more other Partners for the transaction. The need to select another Partner and the number of Partners to select can be determined by various means, according to the implementation of Proof of Justice, such as the Requester Level, a Proof of Justice parameter, a ratio of detected malicious activity increase in the network established by analyzing the number of punished entities over a given number of blocks, a recommendation by one or more Partners, and/or the like. In one or more implementations, the need for one or more Partners is included by the Requester in M1.

According to step 210, in a given implementation, the Requester would then, similarly to step 202, select another Partner (PaN+1). In another implementation, PaN (where Pa1 is for the first iteration) would be the one selecting PaN+1 instead of the Requester. This can be done as a mean of additional resilience against malicious activities which could be attempted by Requesters in the network and

© BB Inventions Inc., 2022

increase (pseudo)randomness entropy during the Block Generation process, which will be described below.

According to step 212, once PaN+1 has been selected, PaN sends a message (MN) to PaN+1, wherein MN contains M1 (or a signed version of PT1, according to the implementation). In one or more implementations, MN would also contain information about previously selected Partners, the number of Partners left to select, the current chaining information (the Partners selection order), and/or the like. Once PaN+1 receives MN, it generates its own Peterson (PN+1) using the abovementioned Peterson -> Generation process. PaN+1 then evaluates MN and determines MN's validity by analyzing it thereof. The analyzing may vary greatly based on the pre-transaction information included therein. In most implementations, one important part of the validation is to evaluate the distance between P1, PN, and PN+1. To do so, PaN+1 would use the abovementioned Peterson -> Tracking process for P1, then determine the distance in blocks between P1 and the block Peterson used in generating PN+1. It would then do the same for PN and PN+1.

According to step 214, PaN+1 then includes PN+1 (and/or PN-X, in one or more implementations) (and P1, in one or more implementations) in a response message (MN+1). In a given implementation of Proof of Justice, MN+1 would also comprise additional information, such as the hash of at least one relevant part of M1 (and/or MN, according to the implementation). MN+1 is then signed and provided back to the Requester (or PaN, which would then provide it to PaN-1 and PaN-1 would compile the information before providing it to PaN-2 and so on until Pa1 is reached, wherein Pa1 would provide the compiled information to the Requester).

Then, step 208 is executed again. If it is determined in step 208 that an additional Partner is required, then steps 210 to 214 are further repeated. Otherwise, step 216 is processed.

According to step 216, the Requester then compiles the information received by all Partners.

<u>Figure 2.</u>

© BB Inventions Inc., 2022

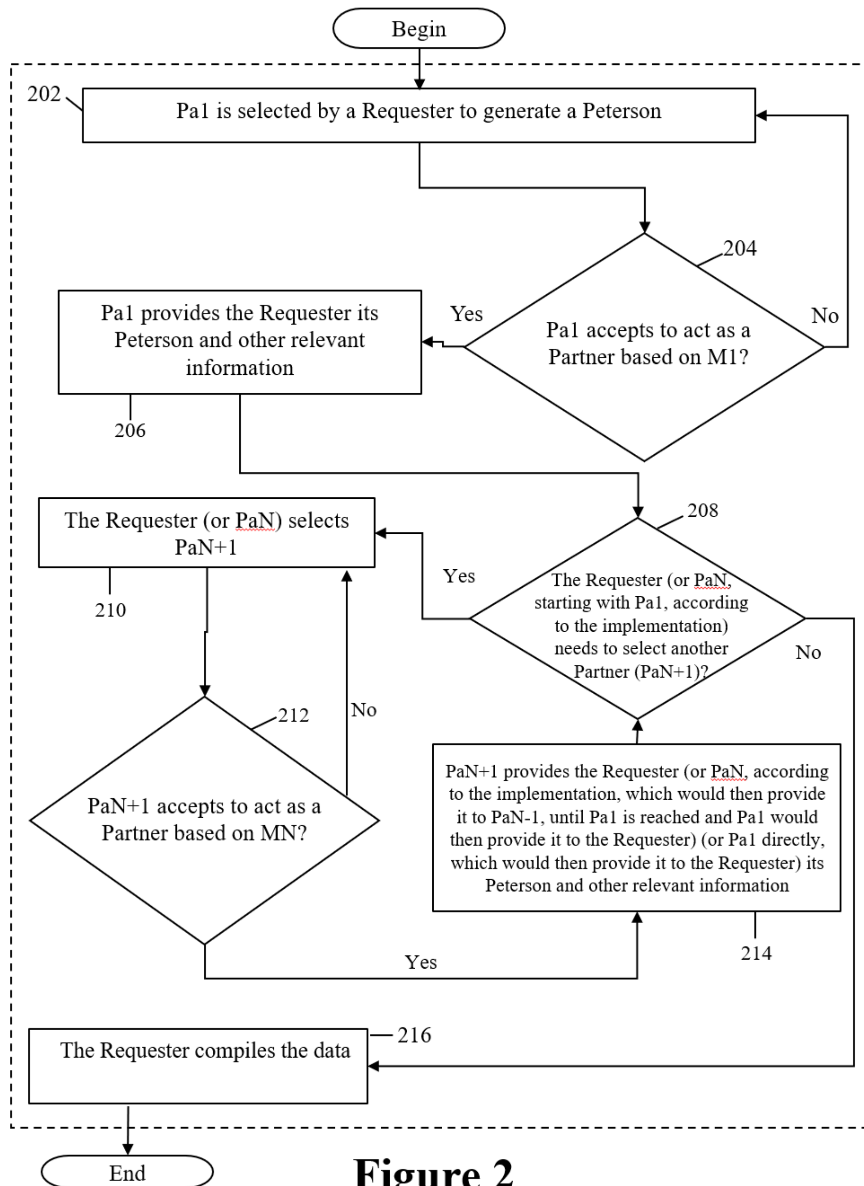


Figure 2

Providing

Once the data has been compiled, it is thereby signed by the Requester and broadcasted throughout the network. The way receiving entities process the transaction will vary greatly based on the Proof of Justice implementation. In one or more implementations, a node receiving a transaction deemed malicious would provide proof thereof to the network using a mechanism similar to the abovementioned "Scoring System -> Entity Reporting System". In one or more implementations, a transaction caching mechanism is used. Said mechanism would essentially allow specific entities with a

© BB Inventions Inc., 2022

“Validator” role to validate a transaction, providing a signed proof of validation to the network and allowing other entities in the network to use said validation verdict instead of validating the transaction itself before adding it to blocks upon generating thereof / their mempool upon receiving thereof. If said one or more of said other entities doesn't trust the result of the verdict of the cached validation, it can validate it manually. Moreover, if after a manual validation the result is not the same as the one cached thereof, proof of such can be sent to the network with a similar system to the one mentioned in the “Scoring System -> Entity Reporting System” section. “Validators” would also get allocated block rewards for their work, much like Speedsters, Enforcers, Generator, and the like. A “lifetime” would also be associated with a given cached transaction verdict. In one or more implementations, the caching is performed by the Partner(s) in the transaction generation process itself if they have the right to do so (i.e., also are associated with the “Validator” role).

Generator & Enforcer

The Generator role is responsible for generating and broadcasting blocks to the network. This role essentially allows a given entity to generate and push the next block, according to multiple criteria. The Enforcer role is responsible for validating parts of a block, specifically the transaction organization, and providing its approval / the differences in the transaction ordering in order to render the block valid.

Criteria & Restrictions

Alongside the default criteria, one additional criterion appropriate for this role could be that the entity must have at least X transactions for each given type, or in total thereof according to the implementation, in its mempool. In one implementation, X could be a Proof of Justice parameter, while in another it could be calculated using a Dynamic Block Size system, which will be elaborated upon below in the "Roles -> Types -> Speedster" section.

Examples of aspects where restrictions would be appropriate to impose for the Requester role could include the number of transactions the Generator can include in a block and/or the number of Enforcers required for said block and/or the acceptance threshold of each given Enforcer for said block could be directly linked to its Generator Level, a sum of blocks generated over time (established using a score, said score being determined using a Long-Term Memory rule, for example), and/or the like.

Pre-Block Information

Pre-block information is data that is required to include in a block in order for said block to be valid. This data is generated and/or obtained prior to doing other block generation processes. Pre-block information comprises things like the Peterson (P1) of the Generator, a subset (which may be the entirety thereof, according to the implementation) of transactions in the Generator's mempool, the hash of the previous block, and any other relevant information.

The number of transactions selected to put in a block can be determined by various means. In a given implementation, the sum of each transaction's size in bytes against a single static maximum block size threshold is the determining factor. In one or more implementations, the single maximum block size threshold is dynamic, and a dynamic minimum block size threshold is also used, based on a Dynamic

© BB Inventions Inc., 2022

Block Size system, said system being elaborated upon below in the "Roles -> Types -> Speedster" section. In one or more implementations, the Generator Level of the entity and/or the number of Enforcer(s) needed for a given to-be-generated block also has an impact on the total number of transactions able to be comprised in said block.

In Proof of Justice, the organizing of transactions in a block by their Partner Peterson (or combination/subset thereof, in case a plurality of Partner Petersons are associated with each transaction) is a crucial component. Indeed, it is part of the reason why transactions have the potential to be feeless in the first place. The ordering can take various forms. For example, ordering the transactions from the smallest associated Partner Peterson to the largest Partner Peterson. It is recommended to use a fast, efficient and reliable self-balancing data structure as a core component of the mempool of each node in order to optimize this process as much as possible. However, several organizing techniques may be used according to the implementation. It is also possible to use several organization techniques and/or multiple data structures, such as one data structure for one or more groups of one or more types of transactions, kind of like by using several tables in an SQL database instead of a table with a "type" column. Using multiple data structures can lead to significant performance gains, according to the implementation of Proof of Justice. It will be understood that as long as the organization method is known (and/or traceable if, for example, in one or more implementations, the ordering of transactions is done using the Peterson of the Generator, although it is recommended to stick to one or more static organization methods throughout the system for performance reasons) by other entities in the network, said method of organization should be possible to implement. Finally, it is important to understand that any organization technique will also need to take into account the nonce associated with each transaction, if and when applicable, on top of the Partner Peterson(s).

In several implementations, once the pre-block information has been gathered, it is then compiled and signed (PT1).

Enforcer Selection

In order for a block to be considered valid in a Proof of Justice-based system, one or more entity(ies) with a role Enforcer (Pa1) must compare the transaction list (T1) of the Generator with its own transaction list (T2), as if Pa1 would generate a block itself, and ultimately provide the comparison result (or proof thereof) (Cr1) to the Generator, said Generator then comprising Cr1 in said block.

It is not recommended that a large quantity of Enforcers verifies and signs every transaction list proposed by every given Generator, as this could lead to severe scalability and performance issues down the line. Therefore, much like Partner(s) for a Requester, a subset of one or more Enforcers must be selected in a determined and traceable fashion for a given Generator, finding an equilibrium between statistical security and performance.

According to step 302 in Figure 3, firstly, Pa1 is selected using the abovementioned Entity Selection process. The Peterson (P1) of the Generator, according to the implementation of Proof of Justice, is also fed to said Entity Selection process.

© BB Inventions Inc., 2022

According to step 304, once Pa1 has been selected, the Generator sends a message (M1) to Pa1, wherein M1 contains PT1. Once Pa1 receives M1, it generates its own Peterson (P2) using the abovementioned Peterson -> Generation process. Pa1 then evaluates M1 and determines M1's validity by analyzing it thereof. The analyzing may vary greatly based on the pre-block information included therein, but will always involve Pa1:

1. Generating its own transaction list (T2) from its own mempool;

2. Making sure T2 is organized in the same way that T1 was organized; and

3. Establishing the difference (or indication/proof thereof) (D1) between T1 and T2.

One important part of the validation that should be in all Proof of Justice implementations is to determine and evaluate the distance between P1 and P2. To do so, Pa1 would use the abovementioned Peterson -> Tracking process for P1, then determine the distance in blocks between P1 and the block Peterson used in generating P2.

According to step 306, Pa1 then includes P2 and D1 in a response message (M2). In a given implementation of Proof of Justice, M2 would also comprise additional information, such as the hash of at least one relevant part of M1. M2 is then signed and provided back to the Generator.

In one or more implementations, a mechanism where the transactions (or indications thereof) in M2 are considered as validated when included by the Generator in the new block (as, in several implementations, the transactions in M2, once included in the block, would be used as reference only and without being considered as validated and confirmed).

According to step 308, it is determined if the Generator (or Pa1 thereof, in one or more implementations) is in need of one or more other Enforcers. Indeed, in one or more implementations, it could be judged that an Enforcer would need to get "Enforced" itself in order to increase security. The need to select another Enforcer and the number of Enforcers to select can be determined by various means, according to the implementation of Proof of Justice, such as the Generator (or Enforcer) Level, a Proof of Justice parameter, a ratio of detected malicious activity increase in the network established by analyzing the number of punished entities over a given number of blocks, a recommendation by one or more Enforcers, and/or the like. In one or more implementations, the need for one or more Enforcers is included by the Generator (or the previous Enforcer thereof, if and when applicable) in M1.

According to step 310, in a given implementation, the Generator would then, similarly to step 302, select another Enforcer (PaN+1). In another implementation, PaN (where Pa1 is for the first iteration) would be the one selecting PaN+1 instead of the Generator. This can be done as a mean of additional resilience against malicious activities which could be attempted by Generators (or previous Enforcer(s) thereof) in the network.

According to step 312, once PaN+1 has been selected, PaN sends a message (MN) to PaN+1, wherein MN contains M1 (or a signed version of PT1, according to the implementation). In one or more implementations, MN would also contain information about previously selected Enforcers, the number of Enforcers left to select for the Generator (and/or one of the previous Enforcer(s) thereof), the current chaining information (the Enforcers selection order) and/or the like. Once PaN+1 receives MN, it generates its own Peterson (PN+1) using the abovementioned Peterson -> Generation process. PaN+1

© BB Inventions Inc., 2022

then evaluates MN and determines MN's validity by analyzing it thereof. The analyzing may vary greatly based on the pre-block information included therein, but will always involve PaN+1:

1. Generating its own transaction list (TN+1) from its own mempool;

2. Making sure TN+1 is organized in the same way that T1 (and TN, if and when applicable, thereof) was organized; and

3. Establishing the difference (or indication/proof thereof) (DN+1) between T1 (or TN thereof) and TN+1. In one or more implementations, instead of comparing T1 (or TN thereof) with TN+1, DN is compared with TN+1 instead. In one or more implementations, the result of a comparison between DN+1 and DN is returned instead of DN+1.

In one or more implementations, if DN+1 contains a number of differences above a determined threshold, the block is considered to be invalid, and the process is thus terminated. It is recommended that a plurality of Enforcers must come to similar a conclusion before terminating the process, in order to prevent possible attacks on the network by Enforcers. In one or more implementations, Enforcers reaching anomalous conclusions are eventually punished by one or more rules.

In one or more implementations, the average of each Enforcer conclusion is considered as the final difference according to T1 and TN+X. In one or more implementations, anomalous conclusions are excluded in the average calculation.

One important part of the validation that should be in all Proof of Justice implementations is to determine and evaluate the distance between P1, PN, and PN+1. To do so, PaN+1 would use the abovementioned Peterson -> Tracking process for P1, then determine the distance in blocks between P1 and the block Peterson used in generating PN+1. It would then do the same for PN and PN+1. It will be understood that the latency between the original M1 request as well as subsequent requests is intended to be considered during the determining of DN+1 thereof. As such, in one or more implementations, a latency threshold of acceptance is used, wherein said threshold can be determined via various means. For instance, it could be calculated:

1. Based on the number of Enforcer(s) to use for a given block;
2. Based on the average overall network latency reported by one or more entities with a given specialized role;
3. Based on a Proof of Justice parameter; and/or

any other calculation deemed relevant for a given implementation of Proof of Justice.

According to step 314, PaN+1 then includes DN+1, PN+1 (and/or PN-X, in one or more implementations) (and P1, in one or more implementations) in a response message (MN+1). In a given implementation of Proof of Justice, MN+1 would also comprise additional information, such as the hash of at least one relevant part of M1 (and/or MN, according to the implementation). MN+1 is then signed and provided back to the Generator (or PaN, which would then provide it to PaN-1 and PaN-1 would compile the information before providing it to PaN-2 and so on until Pa1 is reached, wherein Pa1 would provide the compiled information to the Generator).

Then, step 308 is executed again. If it is determined in step 308 that an additional Enforcer is required, then steps 310 to 314 are further repeated. Otherwise, step 316 is processed.

© BB Inventions Inc., 2022

According to step 316, the Generator then compiles the information received by all Enforcers.

Figure 3.

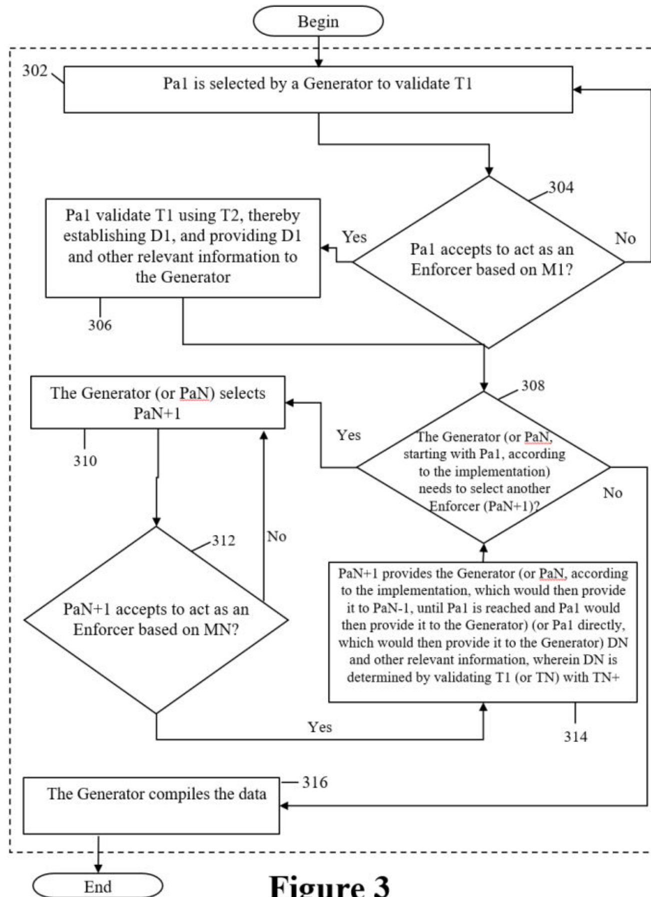


Figure 3

Providing

Using the compiled information from the Enforcers, the Generator finishes the generation of the block, thereby forming the next block (B1). B1 is then pushed to the network.

Cross-Validating

Upon receiving B1, a given receiving entity (R1) will validate B1 and its integrity.

First, R1 will check to see if B1's core architecture is valid. This will involve checking things like its header, the structure of each section of B1, and the like.

© BB Inventions Inc., 2022

In one or more implementations where a Dynamic Block Size mechanism is used (elaborated upon in the "Roles -> Types -> Speedster" section), the checks would further involve validating the size thereof.

Then, R1 will check to see if the transactions in B1 are valid and that there is no conflict with the previous block(s) in the blockchain, that the transactions are organized properly, that all of the signatures are valid, that the Enforcer(s) validation(s) is valid, and so on. The validation of transactions may either be done in full by each and every entity, or with the help of a caching system such as the abovementioned one. The following, in order, will be evaluated:

1. The longest chain will be declared the current one. In the case of equality, go to the next criterion.

2. Compare the score of each Generator(s), find which one(s) has the highest. In the case of equality, go to the next criterion.

3. Compare the Petersons of each chain, and find which one has the most standard distribution of Petersons. In the case of equality, go to the next criterion.

4. Compare the ID of each Enforcer(s), and find the one with the smallest absolute ID In the case of equality, go to the next criterion.

5. Compare the ID of each Generator(s), and find the one with the smallest absolute ID.

Several other criteria may be used/changed, according to the implementation.

Local Database(s) Updates

R1 will then consider B1's chain (BC1) as the current one and update its local chain accordingly.

R1 would then proceed to execute and apply the Rules on BC1, according to the abovementioned "Scoring System -> Applying of Rules" section. Upon applying those rules, R1 would then make sure that the score(s) of each entity part of the network is up to date in its local database.

R1 might also update its local balances of each currency involved in BC1, according to the transactions made in BC1 as well as the block rewards distribution (Elaborated upon below in the Dynamic Block Rewards section).

The end goal is for all entities to maintain a local version of the blockchain and database that is always up to date with the global state of the blockchain. All entities would then use their local version of the blockchain and database to make sure that everything is running smoothly and as intended, and as a reference for any and all relevant processes.

Dynamic Block Rewards

The block rewards are given to the entities according to the roles they have fulfilled within said block, each role having a certain weight over the total block rewards.

The weight attributed to each role is as follows: Storer(s) gets 15%, the Enforcer(s) gets 35%, the Generator(s) gets 30%, the Speedster(s) gets 15% and the Partner(s) gets 5%.

© BB Inventions Inc., 2022

These weights may be changed through a democratically held vote, through slashing mechanisms, or other means according to the implementation of Proof of Justice. An example of a slashing mechanism could be something of the like: For every N number of blocks, the block rewards are halved by X%, calculated according to the number of available rewards. Indeed, the number of available rewards can increase if a "Burn" mechanism is used, such as the abovementioned one in "Roles -> Granting Types -> Staking".

The block rewards may change greatly in nature according to the implementation of Proof of Justice. In one or more implementations, the block rewards consist only of the core coin of the blockchain. In one or more other implementations, each token created on a blockchain using Proof of Justice would be included in the block rewards according to one or more criteria, such as the number of transactions in a given block involving said tokens. Here is an example:

There are 5 transactions in a given block. If the block is filled with:

- 1 Transaction of coin A;
- 2 transactions of token B; and
- 2 transactions of token C,

then the block rewards would be:

- (1/5 * 10) coin A;
- (2/5 * 10) token B; and
- (2/5 * 10) token B,

given that, if all 5 transactions in said block would be consisting of coin A, 10 of coin A would be allocated as block rewards. This technique incentivize competition between tokens in a given Proof of Justice-based blockchain, thereby incentivizing the creator of each token to market said token in order to attract more users, thereby creating publicity for the Proof of Justice-based blockchain. Moreover, such a system enables to delay, in a case where coin A would have a maximum supply (although not recommended in order to conserve the feeless aspect of transactions enabled Proof of Justice), the time where the coin A's maximum supply is reached.

In one or more implementations, every token created on a Proof of Justice-based blockchain would be used as block rewards, without doing the transaction weighting mechanism such as in the previous example.

In one or more implementations, only the tokens would be subject to the transaction weighting mechanism, while the native coin(s) would always be rewarded in full capacity.

In one or more implementations, the reward per token is defined upon the creation of said token thereof and can be updated in a decentralized way, such as a voting mechanism similar to the abovementioned rule updating mechanism in "Scoring System -> Adding, Modifying and Removing Rules -> Modifying a Rule".

In one or more implementations where one or more types of transactions in B1 would have one or more fees attached, said fees would be either:

1. added to the rewards to distribute; or

© BB Inventions Inc., 2022

2. distributed to specific entities based on the transaction type. Indeed, a transaction involving one or more entities with the Code Reviewer role could allocate a great percentage of the fees to the concerned Code Reviewer(s), regardless of the default percentage allocation for block rewards.

In one or more implementations where a Dynamic Block Size mechanism is used (elaborated upon in the "Roles -> Types -> Speedster" section), according to the implementation of such mechanism, the rewards allocation to one or more entities could be affected.

Speedster

The Speedster role is responsible for maintaining ideal block size thresholds according to the current network activity and current network nodes' hardware. This role essentially allows a given entity to create and push a poll aiming at altering one or more block size thresholds as well as vote in polls started by other Speedster(s), according to multiple criteria.

It will be understood that the larger a given block size is, the more demanding a given evaluating and/or generating and/or providing and/or obtaining and/or processing of a given block often is, and high-performance hardware is thereby often required to make the process fast and efficient. Since high-performance hardware is generally expensive and limited, thereby making it inaccessible to some individuals, such as unprivileged individuals, a block size is often directly correlated with the centralization of the blockchain toward stronger computing nodes. On the other hand, a smaller block size often limits a scalability aspect of a given blockchain. Oftentimes, prior art blockchains select the block size to find an equilibrium between maximizing decentralization and minimizing scalability. Yet, such static block size presents drawbacks, such as drawbacks brought in during and as a result of multiple heated Bitcoin™ debates about block sizes, that Proof of Justice's Dynamic Block Sizes can overcome.

Criteria & Restrictions

Alongside the default criteria, one additional criterion appropriate for this role could be that the entity must have been mostly constantly online and reactive for a determined timeframe and have fairly quick node hardware. Another criterion could be to have an average global score above a determined threshold and to never have been blacklisted.

Examples of aspects where restrictions would be appropriate to impose for the Speedster role could include the number of polls the Speedster can start over a determined period of time according to its Speedster Level, the number of polls started by the Speedster / voted in by the Speedster over time (established using a score, said score being determined using a Long-Term Memory rule, for example), and/or the like.

Thresholds

Thresholds define the minimum or the maximum number of transactions, approximately, that can be put in blocks by Generators. The number of thresholds can vary greatly depending on the Proof of

© BB Inventions Inc., 2022

Justice implementation. A block can comprise multiple thresholds. In most implementations, all thresholds are dynamic. In other implementations, only some thresholds are dynamic, while one or more others are static.

Calculation of New Threshold Value(s)

The new value(s) of the block size threshold(s) can be calculated and determined by various means. In a given implementation, the difference over a determined period of time of new transactions of one or more types incoming in the Speedster's mempool (and, in some implementations, a plurality thereof) is the determining factor. In one or more implementations, the score(s) associated with the Requester and/or Partner of each new transaction has an impact on the calculation. For example, several new transactions from untrustworthy entities may not be considered in the calculation. In one or more implementations, the quantity of transactions within a block is correlated with an increase of block rewards, up to a determined amount.

In one or more implementations, the Speedster Level of the entity also has an impact on the validity and acceptability of a threshold update poll. For example, a given Speedster with a mediocre Speedster Level would take the chance of being punished more severely than another, more reliable, Speedster if its poll is rejected by other Speedsters.

In implementations of Proof of Justice using Dynamic Block Sizes, the size of a given block is mainly calculated using the sum of each Partner Peterson per transaction in a given block. In a case of a plurality of Partner Petersons per transaction, a combination thereof is made before calculation. In some implementations where several sections of different transaction types are present in a given block, a weight is allocated to each section, thereby adjusting the resulting sum of Partner Petersons.

In one or more implementations, a given threshold value can only be updated within determined minimum and maximum values.

Threshold Implementation Types

Thresholds can be handled in various ways.

In some implementations, a given threshold is absolute: a Generator cannot infringe a given threshold and is bound by it.

In other implementations, a given threshold is a soft limit: a Generator may infringe it, but according to one or more criteria. For example, a Generator may only be able to do so if its Generator Level is above a determined number. In one or more implementations, if a Generator chooses to infringe the threshold, it will get exponentially diminishing block rewards by doing so. Thus, in a system using conditional (or full) transaction fees, it may be beneficial for a Generator to sacrifice a bit of block rewards to gain more rewards via the transaction fees. It is strongly recommended for implementations using something of the like, however, to implement an additional hard threshold to prevent anomalously large blocks. In some implementations, a Generator is rewarded additional rewards if the block size is on or very close to the soft threshold. Accordingly, in one or more implementations, a

© BB Inventions Inc., 2022

Generator is punished the farther the block size is from the soft threshold, such as significantly lower thereof.

In one or more implementations, a given Generator may be allowed further liberty according to the block size if, for example, said Generator put assets as collateral when generating said block until said block is successfully validated by the network.

In one or more implementations, if a given Generator wants further liberty according to the block size, said Generator could be asked to complete a challenge more or less complex, according to its asks, beforehand.

Upper Threshold

An upper threshold is the maximum number of transactions a Generator is allowed to put in a block. An upper threshold may be either a soft or hard limit, and there may be multiple thresholds according to the implementations (Ex. one soft threshold and one hard threshold). The goal of an upper threshold is to prevent extremely large blocks, thereby preventing or severely limiting most nodes with regular hardware in the network to process said block, centralizing the network, without severely limiting Generators in the block size they can choose.

Lower Threshold

A lower threshold is the minimum number of transactions a Generator is allowed to put in a block. A lower threshold may be either a soft or hard limit, and there may be multiple thresholds according to the implementations (Ex. one hard threshold and one soft threshold). The goal of a lower threshold is to prevent Generators to send blocks with little to no transactions in them as an attempt to attack the network. Moreover, a lower threshold allows a relative decentralization of the network the smaller the threshold value is, giving the opportunity to nodes with less powerful computer hardware to participate in the network.

Pre-Poll Information

Pre-poll information is data that is required to include in a poll in order for said poll to be valid. This data is generated and/or obtained before doing other poll generation processes. Pre-poll information comprises things like the Peterson of the Speedster, an indication of the overall distribution of nodes hardware in the network, the new value(s) of the block size threshold(s) (and, in some implementations, a proof of the calculation behind it), and any other relevant information.

Poll Creation & Voting

Once a Speedster has generated the pre-poll information, it will then sign it and create the poll transaction. The Speedster then broadcasts the poll to the network. In some implementations, creating such a poll requires the Speedster a transaction fee (and/or collateral which will then be returned to the Speedster if the poll is successful). It will be understood that, since the poll is a transaction, one or more

© BB Inventions Inc., 2022

Partners are required, wherein the Partner selection process can be found under the “Roles -> Types -> Requester & Partner -> Partner Selection” section.

In one or more implementations, if two or more Speedsters try to create a poll in a small period of time then one of the two polls will be discarded. As such, there can be only one Speedster poll per block. Several methods can be used to determine which poll will be chosen. For example, a given poll may have more priority over another if the Speedster Level of its creator is higher than the other(s).

Upon validating the poll creation transaction, other Speedsters in the network will vote whether or not they are in favor of altering the threshold(s) by analyzing the pre-poll information specified in the poll creation transaction. For each Speedster, once it established its verdict, it will create a vote transaction indicating as such and broadcast it to the network to be put in a new block. In one or more implementations, a Long-Term Memory rule analyzing if a given Speedster is consistently declining changes to the threshold value(s) from a plurality of reliable Speedsters will see its score(s) negatively impacted.

Once a Speedster sees that the poll is closed (i.e., enough votes have been cast and/or a determined number of blocks were generated since the validation of the poll creation transaction), it will verify the result of the poll and, if accepted, alter the threshold value(s) in its local caching database accordingly.

In several implementations, if said poll is refused, the given block triggering the end event of said poll, the Speedster which created the poll is punished. The nature of the punishment can take various forms. For example, in case a collateral approach is used when creating the poll, the Speedster would thereby lose its collateral, wherein said losing involves either redistributing the full (or a portion of the) collateral to entities in the network and/or reallocating it as block rewards. In one or more implementations, the punishing is proportional to the score(s) of the Speedster. In one or more implementations, Speedsters who voted in disfavor of the result of the poll are also punished. It will be understood that the punishment can vary greatly. In one or more implementations, a mechanism similar to the "Scoring System -> Entity Reporting System" would also be used, allowing entities to report malicious polls even after the fact and punishing / rewarding entities accordingly. For example, if a given poll result ends up being positive but an entity later proves that said poll was malicious, such as by identifying malicious patterns prior and/or during the poll that weren’t identified at the time due to limitation of rules, the malicious entities could suffer slashes, such as elaborated upon in the “Roles -> Special Types -> Blacklisted” section, while non-malicious entities which have been previously punished due to being in non-favor of the poll result would thereby be compensated, such as by seeing their score(s) increase and be allocated an additional weight during block rewards calculation for a determined number of blocks.

Sentinel

The Sentinel role is responsible for monitoring, modifying, or terminating one or more non-deterministic rules (such as AI-related algorithms, machine learning algorithms, etc.). This role essentially involves determining abnormal behavior of one or more non-deterministic rules and acting accordingly in order to stabilize them. Indeed, although helpful in identifying and taking actions against unpredicted potentially malicious actions by one or more entities, a non-deterministic algorithm isn't perfect and may punish entities that don't deserve it. A small error margin is fine, but if the non-deterministic

© BB Inventions Inc., 2022

algorithm starts acting erratically and was to severely impact the network behavior and security, measures have to be taken to maintain stability.

Criteria & Restrictions

Alongside the default criteria, one additional criterion appropriate for this role could be that the entity must have been mostly constantly online and reactive for a determined timeframe and have fairly quick node hardware. Another criterion could be to have an average global score above a determined threshold and to never have been blacklisted.

Examples of aspects where restrictions would be appropriate to impose for the Sentinel role could include the number of polls the Sentinel can start over a determined period of time according to its Sentinel Level, the number of polls started by the Sentinel / voted in by the Sentinel over time (established using a score, said score being determined using a Long-Term Memory rule, for example), and/or the like.

Pre-Poll Information

Pre-poll information is data that is required to include in a poll in order for said poll to be valid. This data is generated and/or obtained before doing other poll generation processes. Pre-poll information comprises things like the Peterson of the Sentinel, the action(s) that the Sentinel deems necessary to take regarding the non-deterministic algorithm, the reasoning behind such action(s), and any other relevant information.

In one or more implementations, if two or more Sentinels try to create a poll in a small period of time then one of the two polls will be discarded. As such, there can be only one Sentinel poll per block. Several methods can be used to determine which poll will be chosen. For example, a given poll may have more priority over another if the Sentinel Level of its creator is higher than the other(s).

In order to determine said action(s), the Sentinel node will analyze the behavior of the non-deterministic algorithm and compare it to a set of weighted parameters in the form of criteria. Here is an exemplary set of criteria that can be used to rationally diagnose and evaluate erratic behavior for a non-deterministic algorithm:

1. Is the algorithm punishing entities for reliable actions covered by already existing deterministic rules?

2. Given that a manual diagnosing process was made by looking at previous verdicts rendered by the non-deterministic algorithm, is the algorithm punishing entities in a way that was not originally intended?

3. Is a second non-deterministic algorithm, or another deterministic rule thereof, predicting that the first non-deterministic algorithm is exhibiting an erratic behavior with a high degree of certainty?

4. Is the erratic behavior of the non-deterministic algorithm having a severe impact on the network?

© BB Inventions Inc., 2022

It will be understood that, in several implementations, most of the criteria will be determined mathematically as an automated process. However, in one or more implementations, as hinted in the exemplary criterion #2, manual processes by users behind entities may also be used and deemed relevant.

If the non-deterministic algorithm is considered to be behaving in an erratic manner, the Sentinel will then calculate and determine what set of action(s) will be necessary to undertake in order to fix said erratic behavior. The list of actions that can be taken is as follow: suspend and activate and update.

Suspend: The Sentinel can temporarily suspend the non-deterministic algorithm in order to give time for a root cause analysis to be made and/or for a new non-deterministic algorithm to be put in place.

Activate: If the non-deterministic algorithm was previously suspended, the Sentinel can reactivate it.

Update: The Sentinel can update the non-deterministic algorithm by changing its parameters, adding new criteria, fixing bugs, and/or the like. This is similar to the abovementioned "Scoring System -> Adding, Modifying and Removing Rules -> Modifying a Rule" section, but specifically for a non-deterministic rule. The rationale behind this is to limit entities that aren't specialized in this technical domain, for example, machine learning, in a case where manual reviews / manual criteria are required in an implementation, to have a biased impact on the network decision, and to leave the decision to specialized entities in the network. This, however, may not be implemented, according to the implementation.

Poll Creation & Voting

Once a Sentinel has generated the pre-poll information, it will then sign it and create the poll transaction. The Sentinel then broadcasts the poll to the network. In most implementations, creating such a poll requires the Sentinel a transaction fee (and/or collateral which will then be returned to the Speedster if the poll is successful). It will be understood that, since the poll is a transaction, one or more Partners are required, wherein the Partner selection process can be found under the “Roles -> Types -> Requester & Partner -> Partner Selection” section.

Upon validating the poll creation transaction, other Sentinels in the network will vote whether or not they are in favor of performing such action(s) on the non-deterministic algorithm by analyzing the pre-poll information specified in the poll creation transaction. For each Sentinel, once it established its verdict, it will create a vote transaction indicating as such and broadcast it to the network to be put in a new block. In one or more implementations, a Long-Term Memory rule analyzing if a given Sentinel is consistently declining changes to one or more non-deterministic algorithms from a plurality of reliable Sentinels will see its score(s) negatively impacted. In one or more implementations, a Sentinel consistently attempting to perform action(s) on one or more non-deterministic algorithms which are denied by the network will see its score(s) negatively impacted.

Once a Sentinel sees that the poll is closed (i.e., enough votes have been cast and/or a determined number of blocks were generated since the validation of the poll creation transaction), it will verify the result of the poll and, if accepted, alter the threshold value(s) in its local caching database accordingly.

© BB Inventions Inc., 2022

In several implementations, if said poll is refused, the given block triggering the end event of said poll, the Sentinel which created the poll is punished. The nature of the punishment can take various forms. For example, in case a collateral approach is used when creating the poll, the Sentinel would thereby lose its collateral, wherein said losing involves either redistributing the full (or a portion of the) collateral to entities in the network and/or reallocating it as block rewards. In one or more implementations, the punishing is proportional to the score(s) of the Sentinel. In one or more implementations, Sentinels who voted in disfavor of the result of the poll are also punished. It will be understood that the punishment can vary greatly. In one or more implementations, a mechanism similar to the "Scoring System -> Entity Reporting System" would also be used, allowing entities to report malicious polls even after the fact and punishing / rewarding entities accordingly. For example, if a given poll result ends up being positive but an entity later proves that said poll was malicious, such as by identifying malicious patterns prior and/or during the poll that weren't identified at the time due to limitation of rules, the malicious entities could suffer slashes, such as elaborated upon in the "Roles -> Special Types -> Blacklisted" section, while non-malicious entities which have been previously punished due to being in non-favor of the poll result would thereby be compensated, such as by seeing their score(s) increase and be allocated an additional weight during block rewards calculation for a determined number of blocks.

Special Types

Full, Lightweight & Storer Roles

For a Proof of Justice-based blockchain, there are usually three types of entities: Full, Lightweight and Storer. A full entity is an entity where its associated node stores the entire blockchain locally (i.e., what people associate with "node" when they hear the word according to blockchain technology). A Lightweight node is an entity where its associated node only stores block headers instead of the full blockchain data and will request additional data from other Storers in the network when necessary. The Storer node is an entity where its associated node is a full node but will also provide historical blockchain data to Lightweight nodes upon request thereof. In order to become a Storer (and, in some implementations, Lightweight), a given entity must meet specific criteria. For example, having enough storage space to store the entire blockchain, having a score above a determined threshold, a reliable connection tested with the help of a challenge-like rule, and the like.

Upon joining the network for the first time, a given entity (E1) will emit a transaction to signal its intention of becoming a given type of node (full, Lightweight, Storer) essentially saying "Hey, I just joined the network, I want to be X". Other entities in the network will then validate if E1 can, in fact, become X. E1 will only have to send this transaction again upon changing X. Otherwise, each time E1 will perform an action in the network, it will be considered that E1 is X. In one or more implementations, entities that don't emit said transaction are considered to be of type full, saving storage and bandwidth by not sending a transaction.

E1 will, from time to time, emit a heartbeat transaction. If after a given amount of time E1 fails to emit another heartbeat transaction, it will be assumed that E1 is offline. In one or more implementations, entities in the network will query E1 from time to time in order to establish if E1 is active. If E1 stays offline for a determined period of time (which can be set as a Proof of Justice parameter), if E1 was a

© BB Inventions Inc., 2022

Storer, a secondary (decentralized) database indicating which node is online and which is not will be updated, allowing other entities in the network can then take E1's place as a Storer (if E1 was a Storer) if they meet the requirements. In most implementations, a full node will automatically become a Storer node if it meets the requirements, without the need to create a transaction indicating as such.

If at any point in time E1 wants to change its node type status (i.e., becoming a full node when it was previously a Lightweight node), it can do so by emitting another transaction with the new desired status. Other entities in the network will again validate if E1 can become the new desired status and update their local states accordingly.

Lightweight / Storer Entities Relationship

A Lightweight node, when it wants to retrieve data from a certain block height H, will randomly select N Storer nodes and send a query to each of them. Storer nodes will answer the queries with the data requested if they have it. If not, they will respond with an error. The Lightweight node will then take the first M responses that were correct (i.e., no errors) and compare them against each other. If all M responses match, the data is considered valid and correct. Otherwise, the process is repeated until either all responses match or there are no more Storer nodes to query.

In some implementations, when a full node wants to become a Storer node, it has to first prove that it is a reliable source of information by responding correctly to Y queries from other entities in the network. In other implementations, this is not necessary.

In some other implementations, Storers will sync each other and sign the blockchain data in a periodic manner in order to standardize the data to eventually provide to Lightweight nodes. This would allow only sending one query to a random Storer node instead of several ones since the data returned by the Storer node would be co-signed, and thus pre-approved, by a plurality of Storers.

Lightweight vs Full vs Storer Entities: Block Rewards Differences

The block rewards that a given entity receives are proportional to the type of entity it is. Full entities will always receive more rewards than Lightweight entities for most, if not all, actions taken in the network for most, if not all, roles. For example, when a full entity generates a new block, acts as an Enforcer, and/or the like, it would receive 90% of its allocated reward(s). The remaining reward(s) for the full entities would then be distributed equally amongst Storer entities. Accordingly, when a Lightweight node generates a new block, acts as an Enforcer, and/or the like, it would only receive 80% of its allocated reward(s). The remaining reward(s) (20%) would then be distributed equally amongst full and/or Storer entities. Accordingly, when a Storer node generates a new block, acts as an Enforcer, and/or the like, it would receive 100% of its allocated reward(s).

The rationale is that entities should be incentivized to take more responsibilities and act as justly as possible in order to earn more.

© BB Inventions Inc., 2022

Blacklisted

An entity is blacklisted when its score(s) falls below a certain threshold. This typically happens when the entity exhibits suspicious behavior. When an entity is blacklisted, it is prohibited from performing certain blockchain-related actions, such as generating blocks, acting as an Enforcer, acting as a Requester, and/or the like.

Blacklisted entities may be subject to slashes, which are a form of penalization. A slash can take various forms and may be different depending on the criterion, blockchain-related event, or blockchain-related feature being considered. For example, a slash may involve prohibiting or limiting an entity from using a loan-related system if one is available, increasing the loan-related interest fee by a determined amount, or any other relevant consequence. In one or more implementations, a slash may take the form of a limitation instead of an interdiction. For example, the quantity of transactions a blacklisted entity is allowed to make for a given number of blocks. In one or more implementations, a slash may take the form of a “burn”, where a portion of a given entity's assets are reallocated to block rewards and/or distributed to the other entities in the network.

In one or more implementations, blacklisted entities are strictly prohibited from performing any blockchain-related action until their score is back over the threshold. In one or more implementations, blacklisted entities are unable to interact with and/or withdraw all or a portion of their assets until their score is back over the threshold. The form that a slash takes may vary greatly depending on the implementation.

An entity may also become blacklisted on a per-role / per-action basis. For example, in one or more implementations, a given entity may become Blacklisted specifically for the Enforcer role, thereby influencing its Enforcer Level, if a Level and XP mechanism is used.

Other factors may also affect a given slash. For example, the number of previous slashes received by an entity, the score associated with the entity, the number of occasions the entity was considered blacklisted, and/or a voting involving at least one entity part of the blockchain may all affect a given slash.

Proof of Justice-Enabled Mechanisms

Parameters

A parameter is a value that is part of the blockchain and may or may not be immutable. A parameter may be updated according to certain criteria, events, or rules. For example, a parameter may be updated based on a voting process involving one or more entities in the network.

A parameter can be updated based on a voting process involving one or more entities in the network. For example, if there is a rule that states that a parameter can only be updated if 2/3 of the entities in the network agree to the update, then the voting process would involve all of the entities in the network. In order for the update to occur, 2/3 or more of the entities would need to vote in favor of the update. The voting process may be initiated by any entity in the network that has the authority to do so. In one or more implementations, only the entities with the highest score(s) can vote in updating a

© BB Inventions Inc., 2022

parameter. In one or more implementations, entities that desire to vote must complete a challenge in order to confirm their reliability. In one or more implementations, the degree of the proposed change(s) to a given parameter impacts the number/score (s) of entities required for the change to take effect. If the change is too important, some implementations may choose to only allow system manager(s) to allow or disallow the change.

Dedicated Node

A dedicated node is a physical device that is used to generate blocks and perform other actions on behalf of an entity in the network. A dedicated node can either be able to perform all or a plurality of roles, or only specific roles. For example, one dedicated node could be able to generate and enforce blocks, while another external device could be able to generate and enforce blocks as well as act as a Storer, having a dedicated hard drive. In one or more implementations, it would also be a requirement in the network instead of an optional thing. The external device is useful because if the phone (for Lightweight nodes) / computer of the user is offline, the dedicated node if it stays in the home of the user, for example, could always be active and participating in the network. Thus, a dedicated node, having dedicated processing power and constant internet access, would definitely be attractive and relevant to users traveling often.

In one or more implementations, every entity in the Proof of Justice-based system must use a dedicated node. The rationale behind this is to ensure that every entity in the system is contributing its processing power and internet connection to the system, and not just relying on other entities. This would help achieve a more egalitarian distribution of resources in the system, as well as ensure that every entity is contributing its fair share.

In one or more implementations, an entity may use multiple dedicated nodes. For example, an entity could have one dedicated node for block generation and a second dedicated node for one or more other role actions. In one or more implementations, an entity may use the same dedicated node for all purposes.

Once the main node, i.e., phone or computer, goes offline, the node cannot receive new blocks or transactions, and thus it cannot validate any data. When this happens, in one or more implementations, a dedicated node would operate while the main node is offline, and once the main node comes back online, the main node will synchronize itself with the dedicated node.

In one or more implementations, entities need to have one or more scores above one or more specific thresholds in order to use one or more types of dedicated nodes.

The use of a dedicated node can help improve the security and performance of the system as a whole, as well as provide a more reliable experience for users. By ensuring that all entities in the system are contributing their resources, it can help to level the playing field and ensure that users are able to participate fairly. Furthermore, by having a physical device that is always online and connected to the network, it can provide a more reliable experience for users, as they will not have to worry about their node going offline and missing out on potential rewards.

Dedicated nodes allow the creator(s) of the Proof of Justice-based system to have an additional, attractive income stream. Since a Proof of Justice-based system node does not require significant

© BB Inventions Inc., 2022

computing power to operate, dedicated nodes can be manufactured at a minimum cost, enabling higher accessibility to users and potentially high-profit margins.

Specialized Subnetworks

In one or more implementations, the network also comprises several subnetworks, each subnetwork being associated with a given role (and/or, in one or more implementations, score(s) associated with entities). For example, there could be a subnetwork for all Generators and another subnetwork for all Enforcers, making it much more efficient to search and find a Generator / an Enforcer. This would help to improve the speed at which data is propagated through the network, as well as reduce the amount of data that needs to be processed by each node.

Proof of Justice - Advantages

A first advantage of Proof of Justice is that it enables to effectively process a transaction without needing to significantly rely on investing resources like PoS, PoW, and PoSpace. Instead, Proof of Justice uses a principle of probabilistic equality in which entities have an equal opportunity to generate the next block given that said entities respect and adhere to a set of determined rules.

A second advantage of Proof of Justice is that it enables the detection of potentially unwanted events and/or the countering of potentially unwanted events by determining a potentially suspicious pattern and thus to create a rule to punish an entity related to the potentially suspicious pattern. For instance, a Sybil attack, which consists of attacking a blockchain by creating a large number of entities to take over decisional-related power of a blockchain, can be countered using one or more rules establishing that a given entity is required to have a certain score to act as an Enforcer, a Generator and/or any other relevant role. Thus, even if an entity makes a Sybil attack without enabling the detection of the potentially unwanted blockchain-related event and/or the countering of the potentially suspicious pattern, the suspicious pattern could still be determinable and penalizable. It will be understood that the potentially suspicious patterns may be provided in various forms. For one or more implementations of a blockchain using PoJ, said blockchain could detect a potentially suspicious pattern selected from a group comprising a rotating and/or an alternating of entities for an Enforcer, a Generator, and/or any other relevant role, ignoring and/or prioritizing the transactions from a similar entity when generating a block, selecting an Enforcer and/or a Partner that was determined in the same timeframe as a Generator, and/or a Requester and/or any other relevant role, determining a transaction of an entity determined in the same timeframe as a Generator when generating a block, determining a transaction of an entity determined in the same timeframe as the majority of other entities when generating a block and/or the like. Therefore, the score associated with an entity performing the aforementioned activities could be impacted negatively when detected, and the entity may be penalized and/or may become blacklisted.

A third advantage of Proof of Justice is that since it enables entities to be evaluated and to detect and/or counter potentially suspicious patterns, attacks such as a 51% attack, i.e., when a majority of entities

© BB Inventions Inc., 2022

attempt to take over a decisional power in the network to perform malicious actions such as maliciously updating a block and/or generating a malicious block in a blockchain and/or generating and/or processing a malicious transaction in a blockchain, can be detectable and possible to fight against by implementing strategies such as:

a) A rule aiming at preventing a given entity to act as an Enforcer, a Generator, and/or any other relevant role by reducing its score when it performs a recognizable and potentially suspicious pattern, thereby reducing the score related to said given entity below a determined region in a score distribution, such as an Enforcer region or a blacklist region;

b) A strategy aiming at enabling sharding, where a given Generator and/or Requester and/or any other relevant role is only authorized to interact with a respective Enforcer and/or a Partner and/or any other relevant role if said respective Enforcer and/or a Partner and/or any other relevant role is comprised in a different shard than said given Generator and/or Requester and/or any other relevant role, the given Generator and/or Requester and/or any other relevant role being associated with said shard based on a given blockchain-related event, the given blockchain-related event being triggered when a given condition is met, such as a determined number of blocks, a determined amount of time elapsed, and/or the like;

c) A strategy aiming at limiting a given entity to repeatedly produce transactions of a given type to a given wallet and/or a given other entity over a determined period of time, thereby limiting said entity from repeatedly interacting with one or more specific entities and/or producing transactions of a given type to one or more specific wallets. For a given implementation of Proof of Justice, the strategy can further comprise requiring an amount of one or more assets comprised in a wallet associated with said given entity to be comprised in a determined range to be able to act as an Enforcer, a Generator and/or any other relevant role, thereby limiting a mass creation of malicious entities. For a given implementation of Proof of Justice, the strategy can also further comprise limiting said entity to interact with other entities associated with wallets comprising a number of assets below a minimum threshold, said minimum threshold being either static or determined based on a plurality of wallets; and/or

d) A strategy aiming at updating, after a determined number of blocks, the score associated with a given entity if said score is over an upper limit in a score distribution in relation to an average score, said average score being based on a plurality of entities, thereby limiting said given entity from repeatedly interacting with a specific Enforcer, Generator and/or any other relevant role. For a given implementation of Proof of Justice, the strategy can comprise temporarily requiring said given entity to act as an Enforcer, a Generator, and/or any other relevant role in relation to a reliability-related event, such as said score updating after the determined number of blocks, until one or more entities can act as Enforcers, Generators and/or any other relevant roles.

© BB Inventions Inc., 2022

It will be understood that because Enforcers are, by default, chosen within a specific score range varying in relation to the number of the Generator, and because suspicious patterns are detectable and punishable, 51% attacks are possible but extremely hard to execute and the malicious entities would be penalized by the determined rules rapidly. Similarly to PoW, PoS, and PoSpace, Proof of Justice is not infallible against 51% attacks, and the latter may succeed in processing fraudulent transactions.

© BB Inventions Inc., 2022