ECASH FOR ESPORTS

Global Cryptocurrency for the eSports Community

Decentralized eSports Ecosystem | 3rd-Party Development Platform

eByte

Technical Paper

With the eByte cryptocoin, digital sport takes a big step closer to real sport.
Introducing the eByte gives rise to new opportunities for which the
eSports Community has been waiting a long time.

AUGUST 2018

TABLE OF CONTENTS

INTRODUCTION

BLOCKCHAIN TECHNOLOGY

Blockchain technology was derived from the first decentralized digital currency, Bitcoin, which was conceptualized by Satoshi Nakamoto in 2008 [1]. Instead of being issued by any institutions, Bitcoin is generated through specific algorithm and massive computing to ensure the consistency of the distributed ledger system. eByte [2] goes further and provides a public blockchain-based computing platform with a Turing-complete language. The core of these cryptocurrency systems represented by Bitcoin and eByte is the underlying blockchain technology. With the components of data encryption, timestamping, distributed consensus, and economic incentives, blockchain technology brings into reality peer-to-peer transactions, coordination and collaboration in a distributed system in which the nodes do not need to trust each other, resolving common issues faced by centralized institutions including high cost, low efficiency and insecure data storage. It should be noted that blockchain technology itself is not a brand new technological innovation, but rather an innovation which combines a series of technologies including peer-to-peer communication, cryptography, blockchain data structures, etc.

Few outside of the gaming industry fully recognize the severity of the player toxicity problem. A vast majority of players contribute positively to gameplay, ensuring fairness, reporting those who cheat, and treating other players with respect. However, a very impactful minority of players oftentimes creates significant problems within a game, and these problems at times, have been so severe within certain games that they've caused significant delays in updates for games. However, the blockchain will be a likely solution for this problem, with the promise that an immutable "reputation scoring" system that uses blockchain smart contracts between players could enable. Problems like cheating and inter-player disrespect could be logged (with each interacting party reporting) and harmful and abusive players could be flagged like never before, since a reputation management system of this type cannot be easily constructed without the fastidious deployment of blockchain technology.

The blockchain enables direct, immutable payment processing between game player and game platform without the use of a third party intermediary. Blockchain will enable very low payment processing fees and incredibly fast speeds, providing players with a faster, cheaper, and more secure means of paying for gameplay and virtual goods. Plus, since all of the payments are on a public encrypted ledger, and since delivery of goods is executed via a smart contract_the incidence of fraud will be reduced tremendously.

The blockchain technology powering the platform ensures that all financial transactions that occur on the platform–from prize money distribution to player salaries–are secure and transparent. All fund-flows are open to the public, but registry information and personal data stays encrypted and secure.

MARKET POTENTIAL

Blockchain will enable a transformation in the market for virtual goods, empowering gaming consumers in an unprecedented fashion. Revenue from eSports, or competitive video gaming, will grow to $700m in 2017, a 41.3% increase from 2016 , according to Newzoo research. The industry is forecast to reach $1.5 billion by 2020. Major investors, high-profile celebrities, big-brand sponsors and major tech companies are banking on eSports' profitable trajectory By 2020, the audience for digital sports is projected to reach 286 million regular viewers and 303 million occasional viewers [3].

As audience enthusiasm for eSports continues to grow, so does the revenue that the games generate through sponsorships and the sale of tickets, merchandise, advertising and media rights. eSports revenue grew by 51.7% between 2015 and 2016—from US$325 million to US$493 million, and was expected to reach almost US$700 million in 2017.

According to current projections, total eSports revenue will more than double again in the next few years, increasing to nearly US$1.5 billion by 2020. Clearly, the eByte project aims to serve a dynamic, prosperous, and rapidly growing market [6]. Global revenue growth of esports can be seen in the Fig 1.

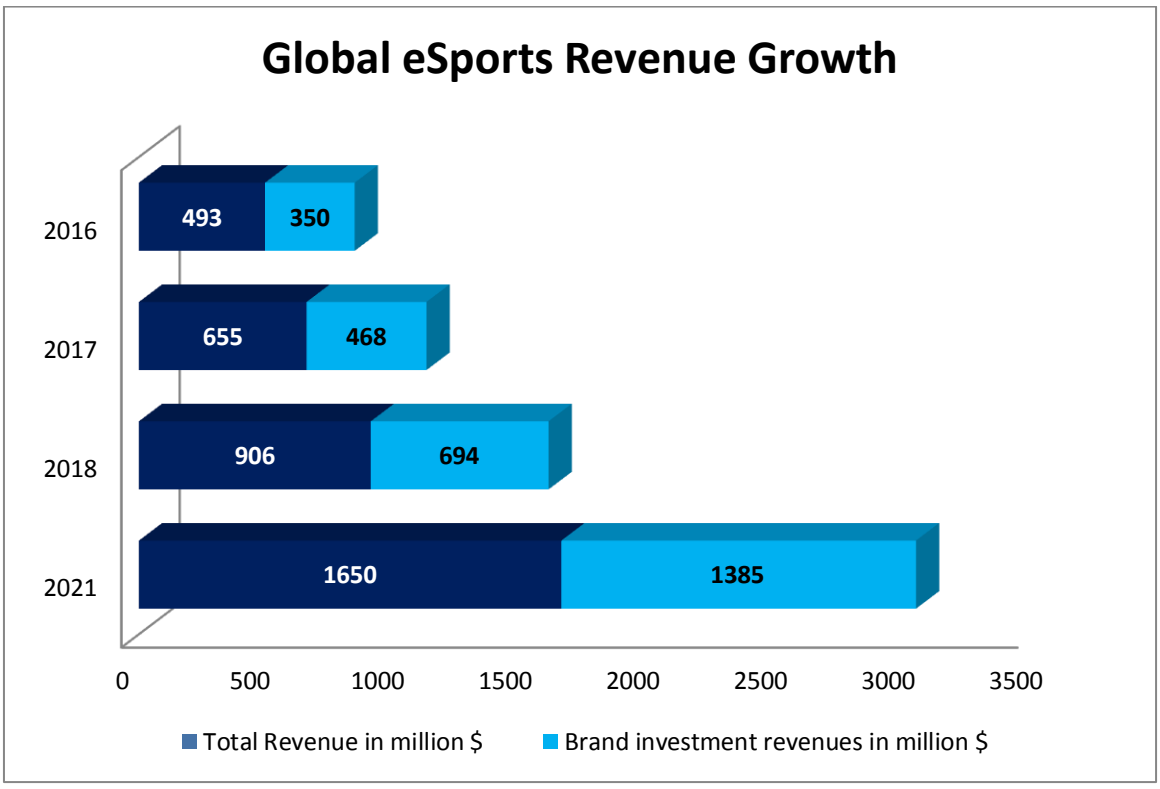

Fig.1: 2016-2021 Global Esports Revenue Growth (Newzoo)

CHALLENGES AND SOLUTION

Despite the amazing market potential and growth eSports currently lacks in possibilities that are fundamental for its future development. Our vision with eByte is to contribute to the maturation of eSports by solving the following challenges.

- **Uniform Portal:** Investment decisions are very difficult since there are no uniform portals that allow uncomplicated and secure participations in eSports. eByte will offer a new platform (the eByte Team Market) that provides future investors and sponsors a complete overview of the market through specific measures and manageable market values of teams, leagues and players.

- **Insufficient financial compensation:** Only high ranking players are rewarded for their performance. This results in inhibited growth of eSports industry. Monetization of eSports projects by secure distribution of participation fees, prize money to all participants and generating income through validation.

- **No Financial Safety:** Players invest a lot of time and energy in training and competitive games but have no financial safety. The eByte smart contracts enable teams to send their players regular compensation via blockchain. Teams are able to bound successful players for longer terms with player contracts.

- **No added value:** Betting on eSports via popular platforms offers no added value for the actual content producers. The eByte blockchain enables sponsors and fans to support their favorites in cryptocurrency in an easy and secure way. Through eByte betting portals the manufacturers of eSports content will then be able to make profits.

- **No Rewards:** Community service providers cannot be rewarded in a fair and easy way for their work. By using the eByte blockchain technology remuneration of community services can be processed securely and transparently in a new way. This will increase trust inside the community and will contribute to establish new jobs in eSports industry.

BUSINESS AND TECHNOLOGY CHALLENGES

As more people join in the development of decentralized self-governing systems, the world has seen a dramatic increase of the number of blockchain projects to over 2,000 projects with the value of global encrypted digital assets amounting to $90 billion dollars. The number of blockchain users and digital asset owners is also rising rapidly from 2 million in the beginning of 2013 to 20 million in early 2017. By 2020, the number of blockchain users & digital asset owners is expected to reach or surpass 200 million, and by 2025, 1 billion [5]. More blockchain-based use cases and applications are emerging with the popularity of blockchain technology. Use cases may have extended with time from digital currency like bitcoin to smart contracts developed by Ethereum, settlement layder developed by Ripple to esports gaming framework like eByte. These diverse application use-cases have also come with increasing demands and challenges on the underlying blockchains themselves.

Measurement of value: One of the core challenges existing blockchains have is lack of a measurement of value. We believe that the blockchain ecosystem needs a universal measurement of value to measure the value of both users and smart contracts. upper-layer applications can be built on this universal measure of value to seek deeper value in its particular use-case. In this sense, innovations in business models will abound in the future, reminiscent of Google's rise in the world of Internet.

Retailing challenge: With so many options of games, connecting fun and challenging games with players who may enjoy them is difficult and expensive. Game developers are required to spend significant sums on marketing and advertising to attract new players.And, to keep players engaged while new games continue to arrive in an endless stream, is a tall order. Statistics show that up to 85% of players abandon a game within the first 30 days.

Application ecosystem construction: With distributed applications (DApps) increasing rapidly on the blockchain, a sound ecosystem becomes the key to better user experience. What major concern is how to help users search and find the desired DApp they need from a massive collection of blockchain applications, how to encourage more developers to develop more DApps for users, and how to assist developers with the construction of DApps. Take eByte for example, hundreds of thousands of apps can be built, however once this increases to the size of applications in the Apple App Store, searching and finding those gaming DApps will be a huge challenge.

DESIGN RATIONALE

Proof of authority algorithm not proof of work

The use of Proof of Work mining, required a certain amount of work to be mined. This allowed users to simply pick the longest valid chain with the highest amount of work as the correct chain. However, Proof of Work is extremely inefficient in terms of energy consumption and latency in verification of transaction and addition of new blocks. This makes it expensive and incentivizes miners to centralize the hashing power. Security of proof of work network depends upon the computing power participating in the network. Distribution of that computing power is another important factor. Another alternative was needed. Proof of authority is more centralized as compared to proof of work, because in order to become the new validator; existing validators vote for the new member. The trusted/added validators can add any block or validate the transaction. We preferred performance upon complete decentralization. Validating process is the most critical to the security of any blockchain network. We prioritized two things, security and performance. Only registered validators can become a part of the network. So attack surface is much shortened. Furthermore, the network can detect malicious validators and deauthorize them by using a proper consensus mechanism. The Block time is very low because the consensus algorithm is not computationally expensive, thus making it possible to achieve rapid transactions. The authority in PoA algorithm is mutual assent to a settlement between the various parties based on the pledge of one set of assets against a whole other set. Thus, if one party falls out of consensus, the other parties automatically assume the non-consenting party's assets and liabilities so that end users remain unaffected. eByte is a community focused trusted network so it is trivial to evaluate, incentivize trusted block producers and penalties malicious block producers are compared to keep control over public decentralized network.

Clique: Proof of Authority

EByte network will use Clique proof of authority protocol that is implemented and being used by ethereum testnet Rinkeby. There is another famous implementation of proof of authority protocol offered by Parity but it seems more complex as compared to clique and not well documented and adopted. Furthermore Clique reduce attack surface by design against known attack vectors by design like malicious signer, signers censoring and concurrent blocks. It may happen that a malicious user gets added to the list of signers, or that a signer key/machine is compromised. In this scenario clique allows a signer to mint merely one block, in order to reduce the damage that potentially can happen.

Governance DApp

In Clique PoA, governance process blocks and manages the singers and govern the consensus securing the blockchain. In this scenario, validators can vote for new signer and blacklist existing validator because of their malicious activity. Security of the network will increase as number of signers increases. Greater the number of signers distributed across the world, greater security of the network. EByte governance rules will be directed with the help of DApp to avoid changes in core proof of authority protocol and it will make possible for anyone to submit proposal. As, customization in core protocol layer will make the network rigid and will provide limited functionalities in accordance to governance of validators and network management, sovereign identities. Thereupon, POA network supports decentralized applications decentralized applications With this abstraction governance is flexible and can be updated independent of clique protocol layer. It will provide interactive user interface integrated in eByte wallet.

EBYTE BLOCKCHAIN

The intent of eByte is to provide transparent and decentralized protocol for esport gaming ecosystem. In regular esports platforms community service providers cannot be rewarded in a fair and easy way for their work. By using the eByte blockchain technology remuneration of community services can be processed securely and transparently in a new way. A lot of people are trying to use blockchain for gaming internal network currency and use it to compensate people or pay for goods and services within the game but the challenge is the scalability. The mature available public blockchain networks mostly works with transaction throughput 10-30 per second. But for planning to have a big gaming platforms like eByte where ecosystem usually yield teams, smart contract based clubs, competitions, marketplace and hundreds of transactions per second; you need to have network with high transaction throughput. In order to solve this high throughput challenge, eByte is using consortium chain. Consequently, scalability depends on hardware used instead of consensus for longest or correct chain. Moreover, blockchain network is scaled vertically and block validators are enforced to use large data centers to process and store this data. Block size will be adjusted according to the capacity of the network so that large number of transactions can be adjusted into a single block. eByte will be a low cost framework of trust for gaming ecosystem where players will be able to become a part of smart contracts based clubs, teams and competitions. Where players can have absolute certainty that their scores cannot be altered at any point at a future date not by a player, not by a hacker, since all of these scores will be kept on an immutable blockchain ledger.

BLOCKS, STATE AND TRANSACTIONS

From a technical standpoint, ledger of eByte blockchain network can be thought as a state transition system where “state” consists of number of transaction performed by that time and “state transition function” that takes a state and transactions and output and new state.

EBYTE ACCOUNTS

The global “shared-state” of eByte is comprised of many small objects i.e. accounts, that are able to interact with one another through a message-passing framework. Each account has a state associated with it and a 20-byte address. An address in Ethereum is a 160-bit identifier that is used to identify any account.

There are two types of accounts:

- Externally owned accounts, which are controlled by private keys and have no code associated with them.
- Contract accounts, which are controlled by their contract code and have code associated with them.

Accounts in eByte network have transaction count and balance maintained as part of the eByte state. These accounts can also have contract code (possibly empty) and storage state (possibly empty) associated to them. Though homogeneous, both accounts can be distinguished: one; account with associated empty contract code (thus the account balance is controlled, if at all, by some external entity) and other with associated non-empty code (thus the account represents an Autonomous Object).

The account state is consisted of four following fields:

Nonce: If the account is an externally owned account, this number represents the number of transactions sent from the account's address. If the account is a contract account, the nonce is the number of contracts created by the account.

Balance: The no. of Wei (sub-denomination, one in which the integer values of currency are counted) owned by this address.

storageRoot: Root node's hash (256-bit) of Trie that encodes the storage contents of the account.

codeHash: The hash of the EVM (eByte Virtual Machine) code of this account. For contract accounts, this is the code that gets hashed and stored as the codeHash. For externally owned accounts, the codeHash field is the hash of the empty string.

An externally owned account has no code or say possibly empty, and one can send messages from an externally owned account by creating and signing a transaction; in a contract account, every time the contract account receives a message it's code activates, allowing it to read and write to internal storage and send other messages or create contracts in turn. Note that "contracts" in eByte should not be seen as something that should be "fulfilled" or "complied with"; rather, they are more like "autonomous agents" that live inside of the eByte execution environment, always executing a specific piece of code when "poked" by a message or transaction, and having direct control over their own eByte balance and their own key/value store to keep track of persistent variables.

EBYTE WORLD STATE

We know that eByte global state consists of a mapping between account addresses and the account states. The mapping between the 160-bit identifiers (addresses) and data structure (account state) is known as eByte world state. This mapping is stored in a data structure known as a Merkle Patricia tree called 'Trie' through the implementation and not burdened on the blockchain itself.

This *trie* entail a simple database backend for the byte arrays' mapping. This mapping database can be named as state database. In this type of data structure, the hash of the root node is cryptographically dependent on all internal data and immune the identity for the whole system state. Being immutable database structure, previous state having known root hash is allowed to be recalled by altering the root hash. These root hashes are stored in the blockchain and can be reverted to old states.

A Merkle tree is a type of binary-tree composed of a set of nodes with:

- a large number of leaf nodes at the bottom of the tree that contain the underlying data
- a set of intermediate nodes, where each node is the hash of its two child nodes
- a single root node, also formed from the hash of its two child node, representing the top of the tree

The data at the bottom of the tree is generated by splitting the data that we want to store into chunks, then splitting the chunks into buckets, and then taking the hash of each bucket and repeating the same process until the total number of hashes remaining becomes only one: the root hash [7].

This data that is being divided into chunks and eventually into buckets can be the data related to the clubs and teams consisting of their players. The tree structure can better be seen in Fig 2.
Every value stored inside the tree is associated with its key. In eByte's case, the key/value mapping for the state tree is between addresses and their associated accounts, including the balance, nonce, codeHash, and storageRoot for each account. The reason this works is because hashes in the Merkle tree propagate upward — if a malicious user attempts to swap a fake transaction into the bottom of a Merkle tree, this change will cause a change in the hash of the node above, which will change the hash of the node above that, and so on, until it eventually changes the root of the tree.

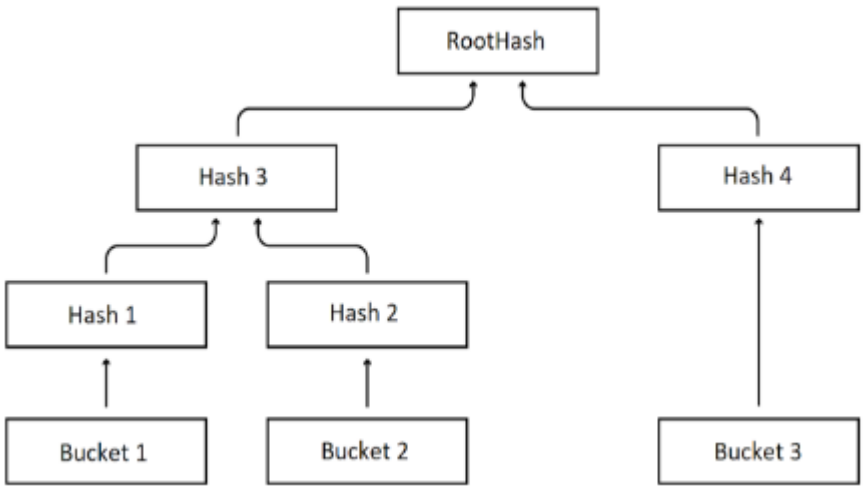

Fig.2: Merkle-Patricia Trie Data-Structure

eByte Gas

One very important concept in eByte is the concept of fees. Every computation that occurs as a result of a transaction on the eByte network sustain a fee. This fee is paid in a denomination called 'gas'. Gas is the unit used to measure the fees required for a particular computation. Gas price is the amount of eByte you are willing to spend on every unit of gas, and is measured in '*gwei*'. *'Wei'* is the smallest unit of Ether, where 1^{018} Wei represents 1 Ether. One gwei is 1,000,000,000 Wei.

With every transaction, a sender sets a gas limit and gas price. The product of gas price and gas limit represents the maximum amount of Wei that the sender is willing to pay for executing a transaction. The sender is refunded for any unused gas at the end of the transaction, exchanged at the original rate.

gasLimit X gasPrice = Transaction fee

In case the sender does not have enough gas, the transaction processing fails and any state changes that occurred are reversed, such that end up back at the state of eByte prior to the transaction. Additionally, a record of the transaction failing gets recorded, showing what transaction was attempted and where it failed. And since the machine already expended effort to run the calculations before running out of gas, logically, none of the gas is refunded to the sender.

All the money spent on gas by the sender is sent to the "beneficiary" address, which is typically the validator's address. Typically, the higher the gas price the sender is willing to pay, the greater the

value the validator derives from the transaction. Thus, the more likely miners will be to select it. In order to guide senders on what gas price to set, validators have the option of advertising the minimum gas price for which they will execute transactions.

Not only is gas used to pay for computation steps, it is also used to pay for storage usage. The total fee for storage is proportional to the smallest multiple of 32 bytes used.

eByte transaction

A transaction is a single cryptographically-signed instruction constructed by the external user in eByte network. That user is assumed to be an actual person in nature. The transactions in ebyte network can be of two types: one that out-turn some message call and other with creation of new account. Both the transactions have some common fields:

- **Nonce:** A value equal to the no. of transactions sent by the sender.
- **gasPrice:** A value equal to the no. of Wei to be paid for all computation costs incurred as a result of the execution of the transaction.
- **gasLimit:** A value equal to the maximum amount that should be used in executing the transaction. This is paid up-front, before any computation is done and may not be increased later.
- **To:** The 160-bit address of the message call's recipient or, for a contract creation transaction.
- **Value:** A value equal to the number of Wei to be transferred to the recipient of message call or, in the case of contract creation, as an endowment to the newly created account.
- **Other values: v, r, s** are some values corresponding to the signature of the transaction and used to determine the sender of the transaction.

Additionally, a contract creation transaction contains:

- **init:** An unlimited size byte array specifying the code for the account initialization procedure. This init is a code fragment it returns the body, a second fragment of code that executes each time the account receives a message call (either through a transaction or due to the internal execution of code). init is executed only once at account creation and gets discarded immediately thereafter.

In contrast, a message call transaction contains:

- **data**: An unlimited size byte array specifying the input data of the message call. The address hash is slightly different: it is either a 20-byte address hash or, in the case of being a contract creation transaction it is empty byte sequence.

Another way to taking, that transactions are what bridge the external world to the internal state of eByte. Messages or internal transactions are considered similar to transactions, with the major difference that they are not generated by externally owned accounts. Instead, they are generated by contracts. This concept can be better understood with the Fig. 3.

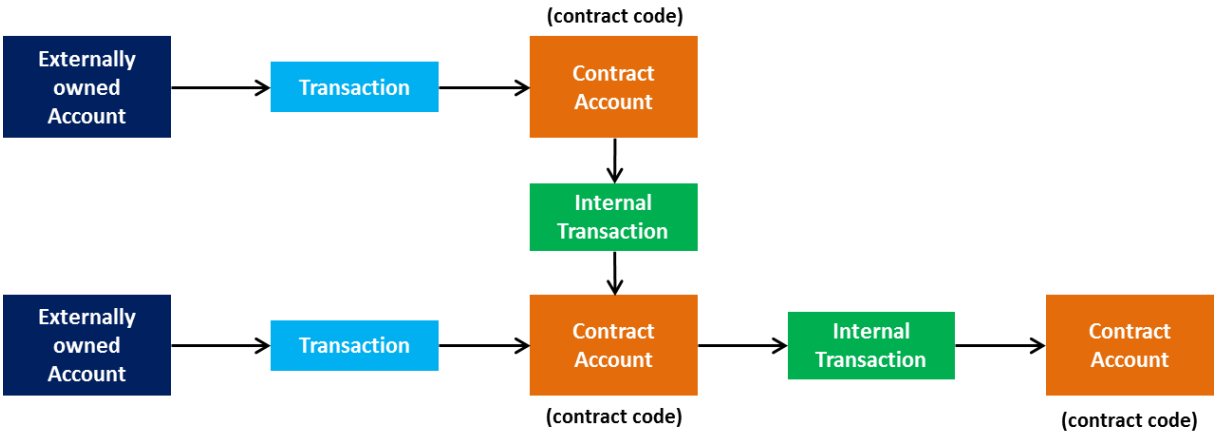

Fig.3: Transactions between externally owned and Contract accounts

eByte Block

The block in eByte blockchain is the collection of relevant pieces of information (known as the block header) denoted as BH, together with information corresponding to the comprised transactions BT. Every block has a "header" which stores the hash of the root node of three different Merkle trie structures, including:

1. State trie
2. Transactions trie
3. Receipts trie

Since the block header includes the root hash of the state, transactions, and receipts trees, any node can validate a small part of state of Ethereum without needing to store the entire state, which can be potentially unbounded in size. The block header contains several pieces of information:

- **parentHash**: The Keccak 256-bit hash of the parent block's header.
- **beneficiary:** The certain amount of reward, that validator will get for validating the block.
- **stateRoot:** The Keccak 256-bit hash of the root node of the state trie, after all transactions are executed and nalisations applied.
- **transactionsRoot:** The Keccak 256-bit hash of the root node of the trie structure populated with each transaction in the transactions list portion of the block.
- **receiptsRoot:** The Keccak 256-bit hash of the root node of the trie structure populated with the receipts of each transaction in the transactions list portion of the block.
- **bloomLog:** The Bloom lter composed from indexable information (logger address and log topics) contained in each log entry from the receipt of each transaction in the transactions list.

- **number**: A scalar value equal to the number of ancestor blocks. The genesis block has a number of zero.
- **gasLimit:** A scalar value equal to the current limit of gas expenditure per block.
- **gasUsed**: A scalar value equal to the total gas used in transactions in the block.
- **timestamp**: A scalar value equal to the reasonable output of Unix's time() at the block's inception.
- **extraData:** An arbitrary byte array containing data relevant to this block. This must be 32 bytes or fewer.
- **mixHash:** A 256-bit hash which, combined with the nonce, proves that a sucient amount of computation has been carried out on this block.
- **nonce**: A 64-bit value which, combined with the mixhash, proves that a sufficient amount of computation has been carried out on this block.

Formally, we can refer to a block B:

$$B=(BH, BT)$$

The information in block header can easily be understood with the following diagram Fig. 4.

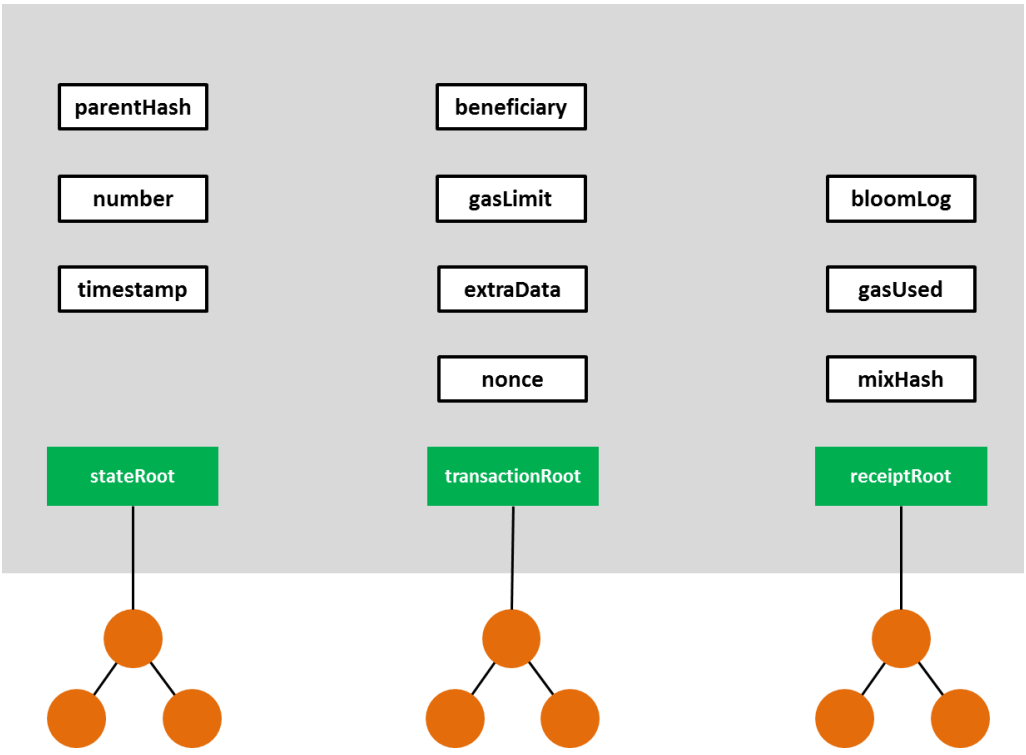

Fig.4: Block Header

Transaction Receipt:

In order to encode information about a transaction concerning which it may be useful to form a zero-knowledge proof, or index and search, we encode a receipt of each transaction containing certain information from concerning its execution. Each receipt, is placed in an index-keyed
trie and the root recorded in the header.

- The transaction receipt is a tuple of four items comprising the cumulative gas used in the block containing the transaction receipt as of immediately after the transaction has happened, the set of logs created through execution of the transaction and the Bloom filter composed from information in those logs and the status code of the transaction that assert to be an integer.
- There is a function that trivially prepares a transaction receipt for being transformed into an RLP-serialised byte array.
- A log entry, is a tuple of the logger's address, a series of 32-byte log topics and some number of bytes of data.
- The Bloom filter function, to reduce a log entry into a single 256-byte hash.

Logs:

eByte allows for logs to make it possible to track various transactions and messages. A contract can explicitly generate a log by defining “events” that it wants to log. A log entry contains:

- the logger’s account address,
- a series of topics that represent various events carried out by this transaction, and
- any data associated with these events.

Logs are stored in a bloom filter, which stores the endless log data in an efficient manner.

eByte Transaction Execution

The execution of a transaction is the most complex part of the eByte network: it denes the state transition function. It is assumed that any transactions executed first pass the initial tests of intrinsic validity. These include:

1. The transaction must be a properly formatted RLP. “RLP” stands for “Recursive Length Prefix” and is a data format used to encode nested arrays of binary data. RLP is the format eByte uses to serialize objects.

2. The transaction is well-formed RLP, with no additional trailing bytes.

3. The transaction signature is valid.

4. The transaction nonce is valid (equivalent to the sender account's current nonce).

5. The gas limit is no smaller than the intrinsic gas, g0, used by the transaction.

6. The sender account balance contains at least the cost, v0, required in up-front payment.

Substate:

Throughout transaction execution, we accrue certain information that is acted upon immediately following the transaction. We call this *transaction substate*, which is a tuple of:

- The self-destruct set: a set of accounts that will be discarded following the transaction's completion.
- Log series: this is a series of archived and indexable *checkpoints* in Virtual Machine (VM) code execution that allow for contract-calls to be easily tracked by onlookers external to the eByte world.
- The set of touched accounts, of which the empty ones are deleted at the end of a transaction.
- The refund balance, increased in order to reset contract storage to zero from some non-zero value. Though not immediately refunded, it is allowed to partially offset the total execution costs.

We define the empty substate A0 to have no self-destructs: no logs, no touched accounts and a zero refund balance.

Execution:

There is a amount of gas the transaction requires to be paid prior to execution. This gas implicitly depends on the series of bytes of the transaction's associated data and initialisation EVM-code, depending on whether the transaction is for contract-creation or message-call.

Some amount say *G* is added, if the transaction is contract-creating, but not if a result of EVM-code. *G* is paid by all contract-creating transactions. Note that the sum of the transaction's gas limit and the gas utilised in the block prior, must be no greater than the block's gasLimit.

The execution of a valid transaction begins with an irrevocable change made to the state: the nonce of the account of the sender, is incremented by one and the balance is reduced by part of the up-front cost. The computation, whether contract creation or a message call, results in an eventual state (which may legally be equivalent to the current state), the change to which is deterministic and never invalid: there can be no invalid transactions from this point.

Execution Model:

The part of the protocol that actually handles processing the transactions is eByte's own virtual machine, known as the eByte Virtual Machine (EVM). The only limitation the EVM has that a typical Turing complete machine does not is that the EVM is intrinsically bound by gas. Thus, the total amount of computation that can be done is intrinsically limited by the amount of gas provided.

Moreover, the EVM has a stack-based architecture. The size of each stack item in the EVM is 256-bit, and the stack has a maximum size of 1024. The EVM has memory, where items are stored as word-addressed byte arrays. Memory is volatile, meaning it is not permanent. The EVM also has storage. Unlike memory, storage is non-volatile and is maintained as part of the system state. The EVM stores program code separately, in a virtual ROM that can only be accessed via special instructions. The EVM also has its own language: 'EVM bytecode'. Fig.5 shows EVM diagram.

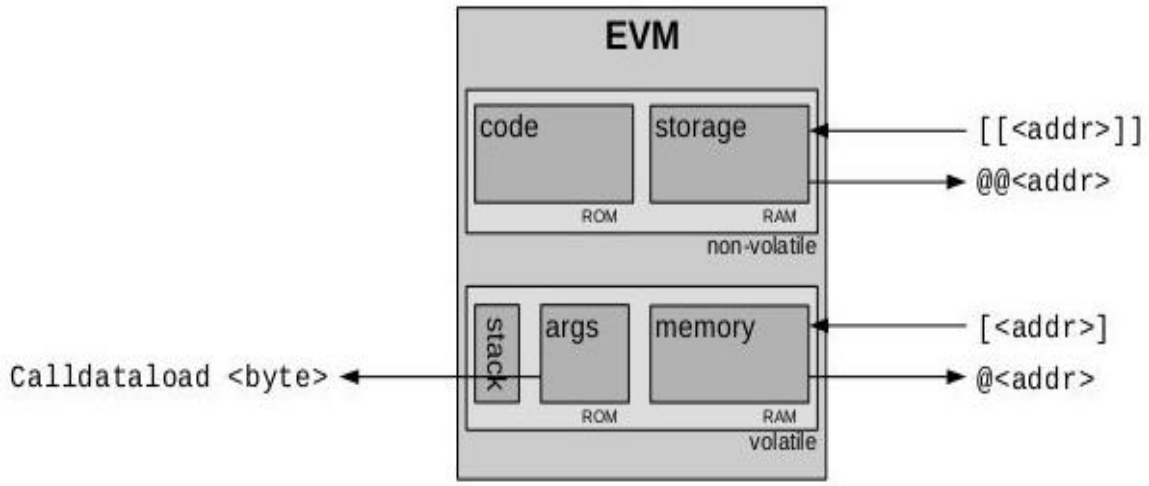

Fig 5: eByte Virtual Machine

Before executing a particular computation, the processor makes sure that the following information is available and valid:

- System state
- Remaining gas for computation
- Address of the account that owns the code that is executing
- Address of the sender of the transaction that originated this execution
- Address of the account that caused the code to execute (could be different from the original sender)
- Gas price of the transaction that originated this execution
- Input data for this execution
- Value (in Wei) passed to this account as part of the current execution
- Machine code to be executed
- Block header of the current block
- Depth of the present message call or contract creation stack

At the start of execution, memory and stack are empty and the program counter is zero. The EVM then executes the transaction recursively, computing the system state and the machine state for each loop.

The machine state is comprised of:

- gas available
- program counter
- memory contents
- active number of words in memory
- stack contents.

On each cycle, the appropriate gas amount is reduced from the remaining gas, and the program counter increments.

At the end of each loop, there are three possibilities:

1. The machine reaches an exceptional state (e.g. insufficient gas, invalid instructions, insufficient stack items and so must be halted, with any changes discarded.

2. The sequence continues to process into the next loop.

3. The machine reaches the end of the execution process.

The block with many transactions get finalized in the end. The block can be new or the existing one. If it's a new block, the process is required for adding this block. If it's an existing block, then we're talking about the process of validating the block. In either case, there are three requirements for a block to get it finalized.

1. *Valida Transactions:* The gasUsed number on the block must be equal to the cumulative gas used by the transactions listed in the block.

2. *Execute Reward:* The beneficiary address is awarded 5 Ether for validating the block.

3. *Verify state and nonce:* Ensure that all transactions and resultant state changes are applied, and then define the new block as the state after the block reward has been applied to the final transaction's resultant state. Verification occurs by checking this final state against the state trie stored in the header.

PROOF OF AUTHORITY CONSENSUS PROTOCOL

In proof of authority, instead of miners racing to find a solution to a difficult problem, authorized signers can create new block, any time at their own discretion. As such, every block (or header) that a client sees can be matched against the list of trusted signers. The protocol of maintaining the list of authorized signers will be fully contained in the block headers so the blocks produced by trusted signers will be become a part of chain.

Authorizing a block

To authorize a block for the network, the signer needs to sign the block's hash containing everything except the signature itself. The means that the hash contains every field of the header (nonce and mixDigest included), and also the extraData with the exception of the 65 byte signature suffix. This hash is signed using the standard secp256k1 curve, and the resulting 65 byte signature (R, S, V, where V is 0 or 1) is embedded into the extraData as the trailing 65 byte suffix. To ensure malicious signers (loss of signing key) cannot wreck havoc in the network, each singer is allowed to sign maximum one out of SIGNER_LIMIT consecutive blocks. The order is not fixed, but in-turn signing weighs more (DIFF_INTURN) than out of turn one (DIFF_NOTURN).

Authorization strategies

As long as signers conform to the above specs, they can authorize and distribute blocks as they see fit. The following suggested strategy will however reduce network traffic and small forks, so it's a suggested feature:

- If a signer is allowed to sign a block (is on the authorized list and didn't sign recently). Calculate the optimal signing time of the next block (parent + BLOCK_PERIOD).
- If the signer is in-turn, wait for the exact time to arrive, sign and broadcast immediately.
- If the signer is out-of-turn, delay signing by rand(SIGNER_COUNT * 500ms).
- This small strategy will ensure that the in-turn signer (whose block weighs more) has a slight advantage to sign and propagate versus the out-of-turn signers. Also the scheme allows a bit of scale with the increase of the number of signers.

Voting on signers

Every epoch transition (genesis block included) acts as a stateless checkpoint, from which capable clients should be able to sync without requiring any previous state. This means epoch headers must not contain votes, all non settled votes are discarded, and tallying starts from scratch.

For all non-epoch transition blocks:

- Signers may cast one vote per own block to propose a change to the authorization list.
- Only the latest proposal per target beneficiary is kept from a single signer.
- Votes are tallied live as the chain progresses (concurrent proposals allowed).
- Proposals reaching majority consensus SIGNER_LIMIT come into effect immediately.
- Invalid proposals are not to be penalized for client implementation simplicity.
- A proposal coming into effect entails discarding all pending votes for that proposal (both for and against) and starting with a clean slate.

Cascading votes

A complex corner case may arise during signer deauthorization. When a previously authorized signer is dropped, the number of signers required to approve a proposal might decrease by one. This might cause one or more pending proposals to reach majority consensus, the execution of which might further cascade into new proposals passing.

Handling this scenario is non obvious when multiple conflicting proposals pass simultaneously (e.g. add a new signer vs. drop an existing one), where the evaluation order might drastically change the outcome of the final authorization list. Since signers may invert their own votes in every block they mint, it's not so obvious which proposal would be "first".

To avoid the pitfalls cascading executions would entail, the Clique proposal explicitly forbids cascading effects. In other words: Only the beneficiary of the current header/vote may be added to/dropped from the authorization list. If that causes other proposals to reach consensus, those will be executed when their respective beneficiaries are "touched" again (given that majority consensus still holds at that point).

Voting strategies

Since the blockchain can have small reorgs, a naive voting mechanism of "cast-and-forget" may not be optimal, since a block containing a singleton vote may not end up on the final chain. A simplistic but working strategy is to allow users to configure "proposals" on the signers (e.g. "add 0x...", "drop 0x..."). The signing code can then pick a random proposal for every block it signs and inject it. This ensures that multiple concurrent proposals as well as reorgs get eventually noted on the chain.

This list may be expired after a certain number of blocks / epochs, but it's important to realize that "seeing" a proposal pass doesn't mean it won't get reorged, so it should not be immediately dropped when the proposal passes.

PROOF OF IDENTITY, INFRASTRUCTURE REQUIREMENTS FOR VALIDATOR

EByte governance rules will be managed with the help of DApp to avoid changes in core proof of authority protocol and it will make possible for anyone to submit proposal. With this abstraction governance is flexible and can be updated independent of clique protocol layer. It will provide interactive user interface integrated in eByte wallet. DApp will accept proposals for adding new signers or removing existing malicious signers. Proposal will be broadcasted as a ballot to all signers so signers will vote and proposal will come into effect is quorum is reached.

DApp functionality will be extended to solve other network related issues with the help of community consensus. Consensus process for adding a new signer will be regulated under *Proof of Identity Protocol.* In PoA, there's only one identity per person, the majority of people only have one true identity. Staking identity means voluntarily disclosing who you are in exchange for the right to validate the blocks. Individuals whose identity (and reputation by extension) is at stake for the securing of a network are incentivized to preserve the network. Clique works on the basis of votes for validating the block and each vote will have some address against it. But only provided addresses information will not be enough to be validated, humans need to provide physical identity of those addresses. So users will submit physical info under proper matrix so existing can evaluate based upon that information.

For the concept to work in real, live settings, a few conditions need to be satisfied:

- Identity must be true: meaning there needs to be a standard and robust process of verifying that validators are indeed who they claim they are.
- Eligibility for bracing identity should be difficult to obtain: so that the right to be a validator becomes earned, valued, and unpleasant to lose.
- The procedure of establishing the authority needs to be the same for all validators: to ensure that the network understands the process and can trust its integrity.

We at eByte PoA Network, has come up with a new approach for the validators of the core network by requiring them to obtain an active public notary license.

To establish that their identity is true: the notaries, who already have their identity information freely accessible in the public domain, go through the formal on-chain identity verification via the PoA network DApps.

As public databases of licensed notaries and the POA Network verification DApps are independent from each other, forging information on either side will prevent a candidate from becoming a validator.

To make eligibility for staking identity hard to obtain: candidates for validators have to overcome the hurdle of passing notary exams. With on-chain governance making it simple to depose a non-conforming validator, losing a validator's role eligibility is public. With their real names at stake, validators are unlikely to act nefariously to threaten their own social standing.

Fulfilling the notary requirements on top of going through the DApps verification process makes the procedure of gaining the reputation/authority explicit and unified and partially independent of the network itself. That establishes integrity and transparency of the process and helps the network participants trust that everyone has the same means to earn the status.

AVAILABLE PROPOSAL TYPES

Add Signer Proposal

Anybody can submit proposal for signer position by using eByte wallet. Proposal will contain user address and its physical identity against that address, existing signers will curate the information critically available at public notary through PoA network Dapps and vote for it. If quorum reached, the signer will be allowed to validate blocks and get reward.

Remove Signer Proposal

If the validator fails to maintain its public reputation and is identified as a malicious signer by eByte monitoring service and service will make a proposal and broadcast to network so signers can verify the report from ebyte blockchain browser and vote for it. If report passed the consensus, malicious signer will be removed and reporter will be get reward from community.

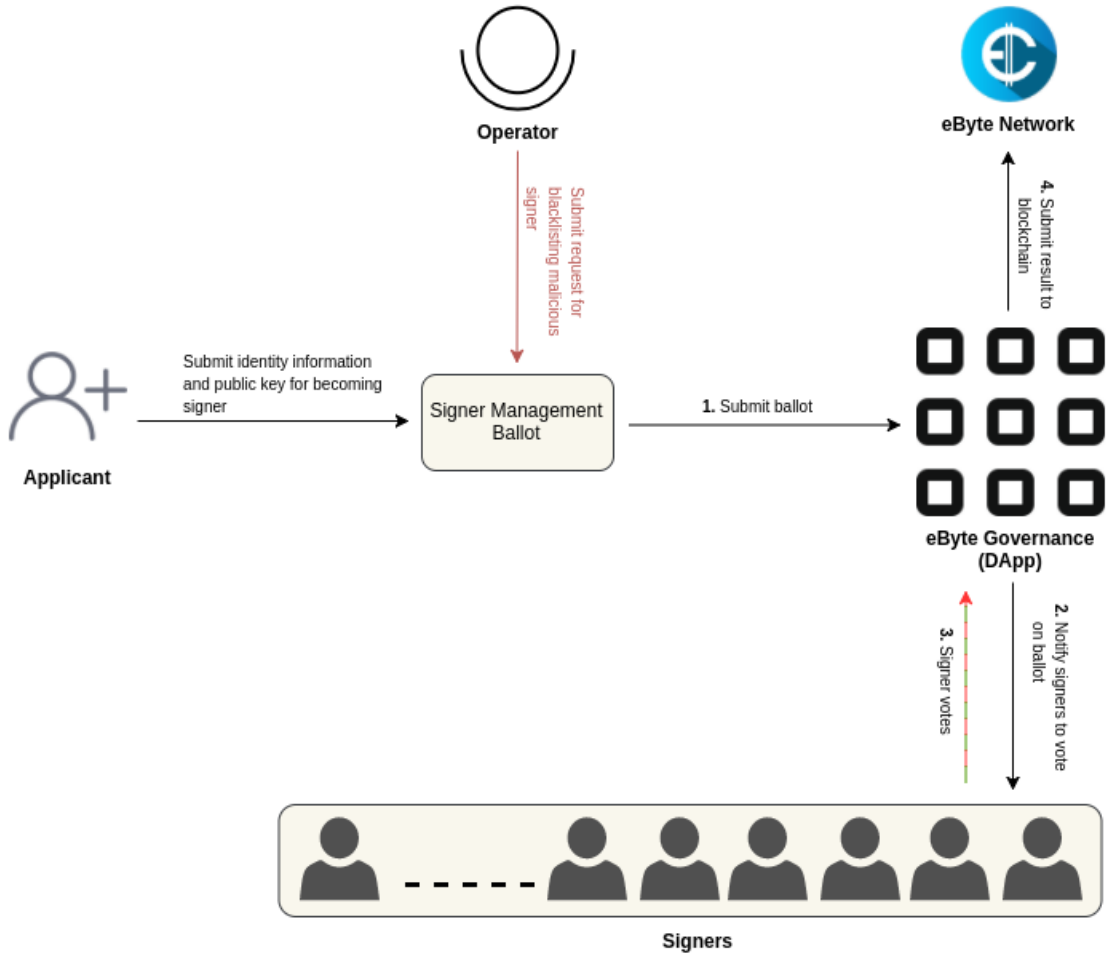

Fig 6: eByte Signer Management

ATTACK VECTORS

An attack vector section section defines the path or means by which a malign person can gain access to our eByte network in order to deliver a payloads or malicious outcome. We also provide the defensive strategies in case of such scenarios. Some possible attack vectors and its their caution has been explained below [3]:

Attack vector: Malicious signer

Malicious user may gets added to the list of signers, or that a signer key/machine is compromised. In such a scenario the protocol needs to be able to defend itself against reorganizations and spamming. The proposed solution is that given a list of N authorized signers, any signer may only mint 1 block out of every K. This ensures that damage is limited, and the remainder of the miners can vote out the malicious user.

Attack vector: Censoring signer

Another interesting attack vector is if a signer (or group of signers) attempts to censor out blocks that vote on removing them from the authorization list. To work around this, we restrict the allowed minting frequency of signers to 1 out of N/2. This ensures that malicious signers need to control at least 51% of signing accounts, at which case it's game over anyway.

Attack vector: Spamming signer

A final small attack vector is that of malicious signers injecting new vote proposals inside every block they mint. Since nodes need to tally up all votes to create the actual list of authorized signers, they need to track all votes through time. Without placing a limit on the vote window, this could grow slowly, yet unbounded. The solution is to place a moving window of W blocks after which votes are considered stale. A sane window might be 1-2 epochs. We'll call this an epoch.

Attack vector: Concurrent blocks

If the number of authorized signers are N, and we allow each signer to mint 1 block out of K, then at any point in time N-K miners are allowed to mint. To avoid these racing for blocks, every signer would add a small random "offset" to the time it releases a new block. This ensures that small forks are rare, but occasionally still happen (as on the main net). If a signer is caught abusing it's authority and causing chaos, it can be voted out.

EBYTE Block Explorer

INTRODUCTION

As the players get increased, clubs become a part of eByte ecosystem and more information will be created potentially. There will be number of tournaments and collaborations occurring on the chain, so ebyte block explorer will crawl the chain to organize the information and make it easily accessible and useful for the users. In eByte ecosystem, all actors will act as a peace of code called '*smart contracts*' without having any functional description attached to it. That's why it will be non-trivial to provide functional description to piece of codes. For effective and meaningful indexing of smart contract eByte team will use following methods:

- In order to provide functional description, crawl the date related to smart contract. It sets up mappings between the data smart contracts of blockchain.
- Explorer will list data under categories like blocks, transactions, verified contracts and unverified smart contracts, clubs,, teams, top players and tournaments.
- Encourage developers to upload verified source code of smart contracts, analyze the functions and semantics of the code, create indexes for the source code, and provide the searching function for similar contracts. For smart contracts without source code, decompile them for their source code.
- Establish standards for smart contracts, so any contract for clubs, tournaments or teams matching those standards can be retrieved and found by users. Also, encourage developers to provide informational descriptions of contracts during smart contract creation.

Search engines have two major functions: crawling and building an index, and providing search users with a ranked list of the eByte network, they've determined are the most relevant. Running a business without search engine is like releasing a film without putting out any trailers. It can help you come up with content for your network, it can show you what people are searching for and make it easy for users to get their desired results.
Our block explorer will help categorize blocks, top clubs, teams, transactions, verified/unverified contracts and list of successful events, most liked teams etc.

INFRASTRUCTURE

eByte planned to develop a centralized search engine for best user experience and performance like etherscan. A dedicated team will develop robust searching service that will crawl and fetch all accounts, transactions and smart contract from eByte ledger. Verifiability of all data from publicly accessible ledger will ensure the impartiality of centralized search engine. Any third party developer will be able to crawl the chain and verify results of eByte search engine. Searching service will retrieve contracts, will parse the code and them categorize them to players, clubs, teams, leagues, championships etc. All data will be available upon ledger so any body can audit the impartiality of our results. It will be possible for any thirty party developer to crawl and verify the data.

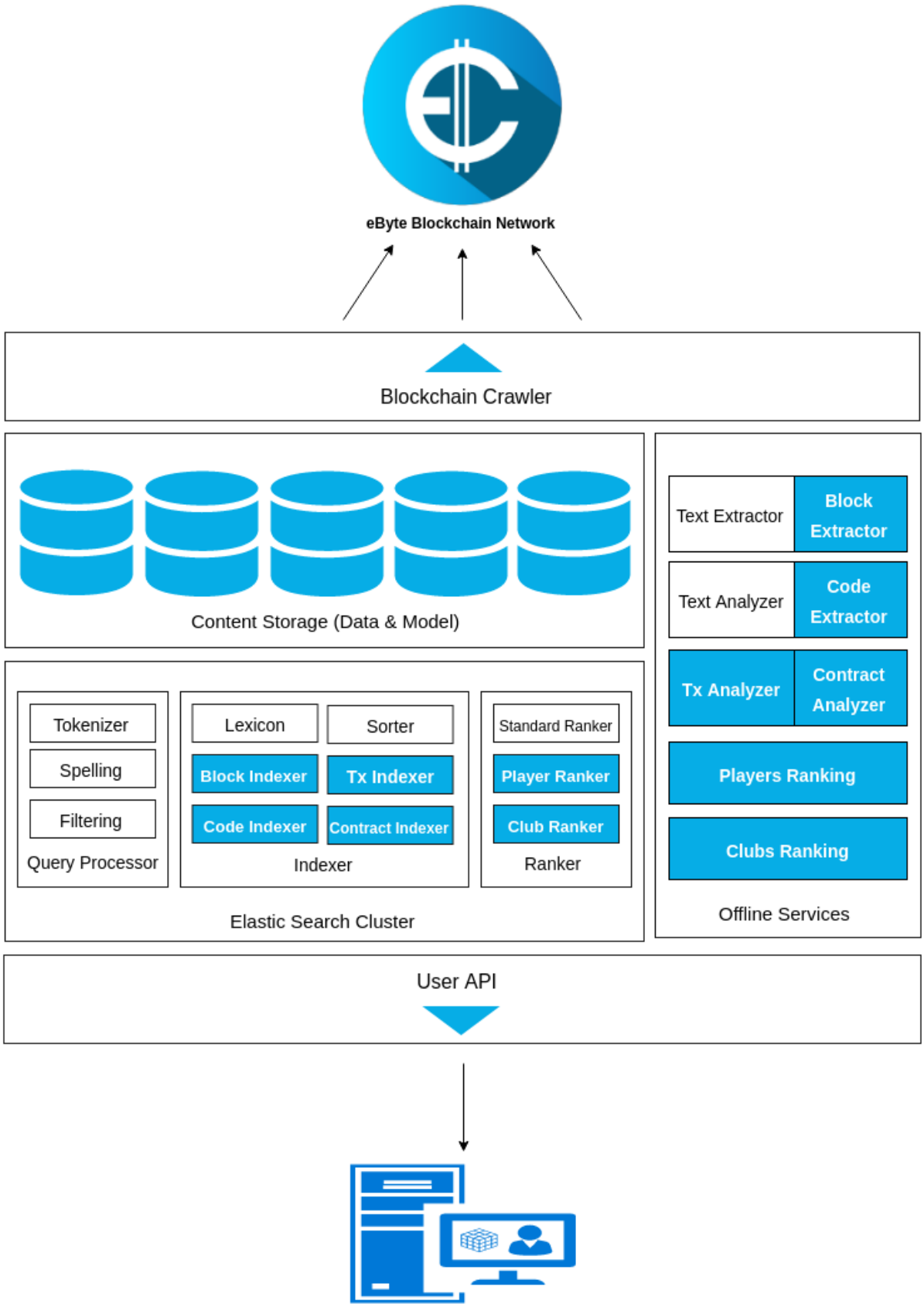

Fig 7: eByte Search Engine

- Crawler

An eByte search engine crawler is a program or automated script that browses the eByte blockchain in a methodical manner in order to provide up to date data to the particular search engine. eByte search engine crawlers are categorized into two types: one for collecting block information and code of smart contracts from blockchains, while the other for crawling data about smart contracts from public URLs including introductions, Dapp user comments and news.

- Tokenizer

When the recipe text is scanned and fed into our search engine, the first thing the search engine has to do is to make sense of the data it receives. We know that words are separated by spaces, and that sentences are separated by punctuation marks. Therefore, computers has to be programmed to break up text into their distinct elements, such as words and sentences. This process is called tokenization and the different chunks, usually words, that constitute the text are called tokens and this is done by our eByte tokenizer.

- Lexicon

eByte lexicon is the special dictionary created based on most searched keywords. It is implemented in two parts: a list of the words and a hash table of pointers. Lexicon can fit in memory for a reasonable price. In the current implementation we can keep the lexicon in memory on a machine with 256 MB of main memory. A program called Lexicon takes the list from sorter together with the lexicon produced by the indexer and generates a new lexicon to be used by the searcher.

- Extractor

eByte extractor is divided into Text Extractor, Block Extractor and Code Extractor, which provide text information, block related information and code extraction service for smart contracts, respectively. It provides easy extraction of keywords from search engine results queries.

- Analyzer

eByte analyzer is consisted of text analyzer, transactions analyzer and contract analyzer, which are text information, block transaction information and smart contract analyzers, respectively. For the smart contract analyzer, it provides contract decompilation, source code extraction, semantic analysis and so on.

- Ranking

eByte ranking tells about the top ranked players and clubs through the list of most favourite players and most participated and winning clubs. In eByte's search engine ranking, in order to rank a club or players with a word query, ranker looks at hit list for those words. Ranker counts the number of hits of each type on the hit list.

- Query Analyzer

It cites to the keyword analysis service, which includes the multilingual word breaker and the spell checker in eByte search engine.

- Indexer

eByte indexer collects, parses, and stores data to facilitate fast and accurate information retrieval. The indexing function is performed by the *indexer* . Indexer reads the blockchain content stored by crawler, uncompresses the content, and parses them. The indexer performs another important function. It parses out all the data in the list and stores important information about them in an anchors file. This file contains enough information to determine where each data points from and to, and the text of the list.

- Sorter

eByte sorter, sorts efficiently in the index database for massive smart contracts and ranking algorithm is used to sort final results. The sorter takes the document, which are sorted by document ID, and resorts them by word ID to generate the inverted index. This is done in place so that little temporary space is needed for this operation. The sorter also produces a list of word IDs and offsets into the inverted index.

- Searcher

eByte searcher is responsible for communicating with the ElasticSearch engine and packing and returning the search result to the searching frontend. Searchers focuses on returning results relevant queries and offers the idea of a "pure" search result which is not influenced by keyword manipulation. It is responsible for communicating with the ElasticSearch cluster and packing and returning the search result to the searching frontend.

- API

API provides external applications with comprehensive searching API services. API services will provide Account APIs, Contracts APIs, Transactions APIs, Block APIs, Event Logs and web socket to facilitate developers to track real time transaction. Ecosystem specific APIs like Club APIs and Player APIs will be also provided.

- Elastic Search

eByte development team uses ElasticSearch cluster for supporting full-text indexing in eByte search engine. Due to lucine integration in elasticsearch, it provides tremendously efficient searching power and being used by number of search engines.

CLUB / TEAM AND COMPETITION LISTS

The eByte network creates the tendency list in combination with listing given by the crawler to provide user visual multidimensional values in blockchains.

Following are some lists produced by out listing algorithms:

- *List of prime clubs*: list of the top ranking, famous clubs which contains best players, winning teams and have most participations.
- *List of top ranking teams:* list of popular teams having max fan following and win majority of the tournaments.
- *List of hyped tournaments:* list of tournaments that have got most highlights in terms of performance and entertainment and much audience has participated in it and created remarkable revenue.
- *Star Players:* list of players who have won matches, made best records and have huge fan following.
- *Validators:* list of claimants who got the validation rights through voting.

KEYWORD QUERY

By providing a keyword or describing the textual information about a smart contract such as its title, author or function, users can find the matching contract from massive smart contracts. Currently, mature and sophisticated algorithms and technologies are available for text searching. By using the natural language processing and inverted index technologies, we can retrieve and sort efficiently in the index database for massive smart contracts. This involves the following key technologies:

1. Distributed crawler for topic oriented technology.
2. Polyglot word breaking technology: word breaking is relatively simple for western words. But problem is with other languages like chinese. For the word breaking of Chinese words, these algorithms are available: positive maximum matching, negative maximum matching, shortest path word breaking and statistical word breaking.
3. Correction for search terms and semantic comprehension.
4. Inverted index and distributed searching architecture.
5. Ranking algorithm for sorting the searched results.

Among these technologies, the algorithm will be designed in combination with eByte crawler and its listing. Specifically, we use intra-user transfers in the blockchain world as an analogy to web page reference relations in the Internet world to create the blockchain transaction graph.

SEARCHING FOR SIMILAR / REDUNDANT

For developers and certain users, they may want to search for smart contracts with similar functions according to the code fragment of a contract. Being different from regular keyword searching, code similarity has its particularity. To implement the searching function for similar smart contracts, we need to use a certain algorithm to measure the code similarity in a number or percentage for club related smart contracts and teams.

In today's technology, some known algorithms for code similarities are available i.e. token sequence similarity, string edit distance, program dependency graph similarity, abstract syntax trees, krinke and abstract syntax tree similarity. In order to find similarities, these code a these algorithms describe the similarity in terms of code text, structure, trees, raphs and syntax from different dimensions. By combining these 6 major algorithms, we put forward several features of the code similarity like signature types, calls, tree skeleton and functions.
Similar to search results of keywords, search results of smart contracts also use the same contract crawling algorithm to sort final results.

eByte DNS

eByte Name Service (ENS) is the service used to convert ambiguous data strings to human readable names. Due to the anonymity of blockchains, account addresses are long and meaningless strings, which are not user-friendly. For this reason, users are prone to misoperations like money loss caused by unintentional funding or the interaction with incorrect objects. In other words, by using domain names that are easy to remember, users can gain better experience. By using smart contracts, the eByte development team will implement a DNS-like domain system named eByte Name Service (ENS) on the chain while ensuring that it is unrestricted, free and open. Any third-party developers can implement their own domain name resolution services independently or based on ENS.
For instance, bob applied for the domain name “bob.ens” for his account address 0xdf4d22611412132d3e9bd322f82e2940674ec1bc03b20e40. To transfer money to Bob, Alice simply needs to enter “bob.ens” in payee information so that the money will be transferred to the correct payee address through the ENS service.
There are some rules for ENS:

- Once the ENS service is active, users can query domain names for availability. For a unoccupied domain name, users can bid for it through a smart contract. The bidding process is open so that any user can query for others’ bids and update their own bids anytime.

- Top-level sub-domain names will be reserved and unavailable for application, such as *.ens, *.com, *.org and *.edu. Thus, users can only apply for second-level sub-domain names.

- Users can transfer the ownership of a domain name with or without compensation. eByte does not intercede in any transactions of domain names.

- Users can surrender the ownership of a domain name at any time. In this case, the bidding funds will be automatically refunded to the user’s account and the domain name will be released as available again after domain data is cleared.

- Once the bidding time-period expires, the highest bidder wins the domain name and the smart contract locks the user’s bidding funds. The validity period of the domain name is one year. After one year, user can freely determine whether to renew or not. If he wants to carry on with the existing name, the validity period will be extended for another year. If want to change the name, the bidding funds will be refunded to the user’s account and the domain name will be released as available to other users.

eByte Utility Coin

eByte token will be known as utility coins because it act as a fuel for eByte blockchain network. Transaction fee will be charged in the form of eBytes utility coins. For the ICO eByte will launch non-perpetual ERC20 eByte tokens upon ethereum network. After the launch of eByte blockchain network those tokens will be converted to eByte utility coins.

Conversion from eByte ERC20 to eByte Utility Coin

eByte blockchain genesis block size will be in hundreds of megabytes with details of all ERC20 ebyte tokens holders. ERC20 token holders will get equal balance of eByte utility coins on the first block of eByte blockchain. So they have to use same wallet to access eByte blockchain in which they have eByte ERC20 token. In theory there will be one to conversion from ERC20 to utility coins. At the moment when eByte blockchain network will be launched, eByte ERC20 token contract will be locked. Just locked!

CONCLUSION

As the popularity of digital sports grows among players and viewers—with matches filling stadiums, streaming live and airing on cable television—eSports is attracting serious attention and support from major brands, real sports organizations, and other investors. Today, eSports is not only a worldwide recreational phenomenon, it is also well on its way to becoming a multibillion-dollar industry.

WHAT WE HOLD?

Our eByte is a blockchain-based, decentralized eSports platform. The eByte project will create an internationally accepted and widely used compensation system for eSports. We have introduced, discussed and formally defined the protocol of eByte. Through this protocol the reader may implement a node on the eByte network and join others in a decentralised secure social operating system. With the development of the eByte blockchain, plus the creation of in-wallet executable smart contracts and a variety of innovative platforms and portals, the eByte project ensures participating leagues, teams and players a fair compensation for their performance and commitment while preserving the decentralized nature of digital sports.
In the distributed economies, for reward distribution blockchain is the transparent and fair mean. For the very first time, blockchain provides the independence to the digital economy monetization system from the hand of third parties like visa, master, paypal etc. OUr blockchain is the consortium blockchain that is ‘trust-based’ and ‘permission-based’ which means they cannot break existing patterns and are considered as improved innovations.

WHAT PROBLEMS ARE BEING SOLVED?

- eByte will offer a new platform (the eByte Team Market) that provides future investors and sponsors a complete overview of the market through specific measures and manageable market values of teams, leagues and players.

- Monetization of eSports projects by secure distribution of participation fees, prize money to all participants and generating income through.

- Validation.

- The eByte smart contracts enable teams to send their players regular compensation via blockchain. Teams are able to bound successful players for longer terms with player contracts.

- The eByte blockchain enables sponsors and fans to support their favorites in cryptocurrency in an easy and secure way. Through eByte betting portals the manufacturers of eSports content will then be able to make profits.

- By using the eByte blockchain technology remuneration of community services can be processed securely and transparently in a new way. This will increase trust inside the community and will contribute to establish new jobs in eSports industry.

FUTURE TRENDS

Partnership between traditional sports franchising and esports franchising is becoming regular day-by-day. Multinational companies like coca-cola, visa and red-bull showing strong interest and commitment in esports community. In the United States, teams like the Philadelphia 76ers and Miami Heat invest in eSports, and European soccer teams like Paris Saint-Germain (France) and Manchester City (UK) have taken players under contract for their representation. Although eSports still is a worldwide recreational phenomenon, its well on it's way to becoming a multi-billion dollar industry soon. Asian olympic Council has also announced esports as official medal game in April 2017, that is going to be held at Hangzhou, China in 2022.

References

[1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008.

[2] V. Buterin, “eByte: a next generation smart contract and decentralized application platform (2013),” URL {http://eByte. org/eByte. html}, 2017.

[3] https://www.forbes.com/sites/outofasia/2017/11/07/how-cryptocurrencies-and-blockchain-are-taking-esports-to-the-next-level/#6eeaeb6a4391

[4] B. Laurie and R. Clayton, “Proof-of-work proves not to work; version 0.2,” in Workshop on Economics and Information, Security, 2004.

[5] Team, Nebula AI. "NEBULA AI (NBAI)—DECENTRALIZED AI BLOCKCHAIN WHITEPAPER." (2018).

[6] https://newzoo.com/key-numbers/

[7] https://chain.com/docs/1.2/protocol/specifications/blockchain

[8] https://github.com/ethereum/EIPs/issues/225

SAVE YOUR

SPOT NOW!

VISIT

EBYTE.SALE