EOS: An Architectural, Performance, and Economic Analysis

Brent Xu[1], Dhruv Luthra[2], Zak Cole[3], Nate Blakely[4]

Abstract— **The EOS system presents a platform for smart contract development and decentralized storage in attempt to address the common scalability issues found in popular blockchain systems such as Ethereum and Bitcoin. Through its implementation of delegated Proof-of-Stake (dPoS) consensus and an architectural design focused on parallel processing and full-duplex communication, the EOS system attempts to provide a decentralized application development environment which offers high transactional throughput. Since the public release of the EOS main-net, multiple parties have speculated and made claims of the systems ability to provide processing capabilities in the range of 5,000 to 1,000,000 transactions per second. However, such performance has yet to be observed in a production network. No comprehensive analysis of the EOS system architecture has been conducted to determine the capabilities of the network.**

The following document outlines the results of a three-month research initiative to analyze the architecture of the EOS system and benchmark its functional performance under a range of environmental conditions. Through practical testing and experiments in a controlled laboratory setting, this research provides a thorough and objective model of its design, performance, and economics in order to present a reference for the blockchain community.

I. INTRODUCTION

The cryptoasset market of 2017 garnered significant mainstream attention with the unprecedented introduction of new users to the blockchain space. This high volume of activity similarly yielded a variety of blockchain-based projects driven by the financial promise of the Initial Coin Offering (ICO). With an easy way to raise the capital needed to develop practical solutions within a new and unregulated market, over 500 total projects have raised an excess of $12 billion to date [1].

One of such projects was EOS, developed to provide an operating system-like construct intended to allow for the rapid scaling of decentralized applications. In order to address the essential shortcomings found within popular blockchain projects, the EOS architecture adopted several novel solutions in order to provide higher processing capabilities and throughput. With a focus on providing an optimized smart contract-oriented platform, much of the message surrounding the EOS ICO positions the system as a competitor and successor to Ethereum, laying the foundation for blockchain 3.0 [2].

The claims of the system's ability to provide transactional throughput in the millions has draw significant attention from the cryptocurrency community. The goal of this research is to validate any speculative claims surrounding EOS and provide a comprehensive understanding of the system design and capabilities.

A. Overview

1) Architecture: In this section, we examine the function and design of the following components that comprise the EOS system:

- Accounts: How users are defined in the EOS network, including the relationship between accounts, wallets, and cryptographic keys.
- Transactions and Contracts: How transactions and contracts are constructed in the network.
- State Management and Blockchain: How the EOS protocol manages the state and resources associated with decentralized applications.
- Execution Environment: How smart contracts and decentralized applications are executed in the EOS network by block producers.
- Consensus: How the EOS network establishes consensus concerning whether or not a transaction, action, or contracts is valid and adheres to the rules of the protocol.

2) Performance: This section presents the results of our performance benchmark testing conducted using the Whiteblock blockchain testing framework in the Whiteblock lab [3]. These tests focused primarily on the following three areas:

Transactional throughput of the EOS network under the presence of various environmental conditions, including transaction volume, network latency, and packet loss.

Fault tolerance of the EOS system, including an analysis of how the system responds to network partitions, forks, and the presence or absence of a varying number of block producers.

Security of the EOS system, including the ability of block producers to manipulate state and the degree of control they can exert over the network.

3) Economic Systems Analysis: This section of the report aims to better understand the economics of the EOS system, such as token design, the behavior it incentivizes in users,

[1]B. Xu is Principal Investigator at Vulcan Labs, ConsenSys. brent.xu@consensys.net
[2]D. Luthra is a blockchain researcher at Vulcan Labs, ConsenSys. dhruv.luthra@consensys.net
[3]Z. Cole is Chief Technology Officer of Whiteblock, a blockchain testing company. zak@whiteblock.io
[4]N. Blakely is a software engineer at Whiteblock. nate@whiteblock.io

and how it effects network function. This section also contextualizes the EOS token sale, provides information about the various utilities of the token, summarizes the token supply, demand, and value drivers, assesses existing game theory of the platform, and presents some potential issues with the EOS economic model.

II. Architecture

Architecture analysis will consist of evaluating the various layers of the platform and identifying the fundamental components that enable its proper function.

A. *Overview*

EOS is a collection of applications that interact within a distributed database structure. As illustrated in Figure 1, the software system is comprised of three primary components: `Nodeos`, `Cleos`, and `Keosd`. Users interact with the network through the command line tool, `Cleos`, which subsequently connects through `Nodeos` to the rest of the database management system (DBMS) [4]. The system is organized as a network of interconnected computational units with a loosely coupled architecture. This allows the replication of processes that enable the system to be updated based on changes to the EOS database while distributing the computational workload among the block producers.

1) General Architecture: Architected as a peer-to-peer blockchain system, EOS utilizes system-optimized approaches to operate in specified execution environments. The system is non-autonomous, based on its consensus model which relies heavily on the 21 block producers. Data and computation are distributed among the 21 block producers in the network creating a central/master DBMS, which coordinates updates across the network. The distributed database is homogeneous since the 21 block producers are processing transactions in order to achieve consensus among all of the separate databases within the EOS network. EOS has created their own LLVM front-end that is used through the EOS C++ tool-chain to compile contracts. The nodes that are implemented should follow the same protocol for processing and validating transactions and contracts across the network [2].

2) Components:

- **Nodeos**: The core component of the node setup is configured as a daemon using various plugins that allow the node to run. Nodes can either be producing nodes (block producers) or non-producing nodes that are only affiliated with an account. Non-producing nodes can be implemented as a public HTTP-RPC API for developers. These nodes can also act as private endpoints for apps that wish to utilize the computational infrastructure for local development [5].
- **Cleos**: Clients can access the EOS network via the `Cleos` command line interface (CLI) tool. This tool allows interaction with the overall network and configuration of interactions within the network through `Nodeos`. The block producers in the network host, deliver, and manage most of the resources consumed by the clients, allowing them to directly access information about transactions [6].
- **Keosd**: This component of the system is designed to store private keys that are used to sign transactions within the network. `Keosd` is run locally and stores keys locally. Users are able to use and query the keys through its RPC API [7].

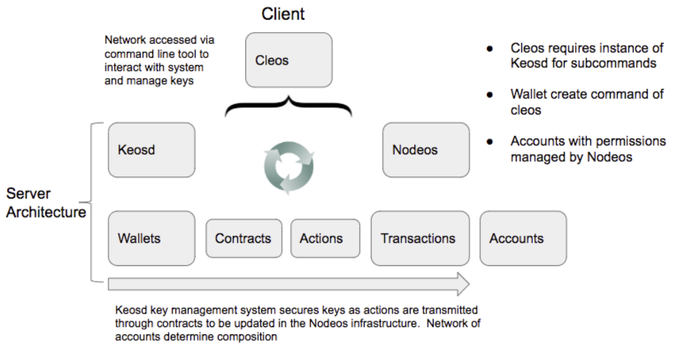

Client
Network accessed via command line tool to interact with system and manage keys
Cleos
Cleos requires instance of Keosd for subcommands
Wallet create command of cleos
Accounts with permissions managed by Nodeos
Server Architecture
Keosd
Nodeos
Wallets
Contracts
Actions
Transactions
Accounts
Keosd key management system secures keys as actions are transmitted through contracts to be updated in the Nodeos infrastructure. Network of accounts determine composition

Fig. 1. Overview of EOS System Architecture

Systems running EOS need pre-built instances of `Cleos` and `Nodeos` that are running at the time prior to running a node in the network [5].

When the network is created, various components allow particular actions to persist throughout the network. These actions accomplish several tasks:

- Loads basic plugins
- Set server address
- Enable CORS
- Adds contract debugging

The network can be considered properly configured when `Nodeos` is validating/syncing blocks in the network. When an alias affiliated with the `Cleos` tool is created on a local system, a command is processed in the `Nodeos` application [7]. This network setup is necessary for proper coordination to exist between the different components of the network. As the interface is created for operation across nodes, the network allows for transactions to be processed efficiently using the shared computational resources of the block producers.

3) Contract Development Toolkit (CDT): The CDT is designed to organize the tools necessary for contract compilation and ABI generation. The binaries associated with `eosio.cdt` are installed on the local machine. The `eosio.cdt` repository contains the `eosio-cpp` component which compiles C++ code to WASM. This is also how ABI's are generated to allow user actions to convert between JSON and their binary form [8].

B. *Accounts*

1) Overview: EOS implements an account-based mechanism for maintaining a record of state. Understanding these accounts will provide an understanding for the system's

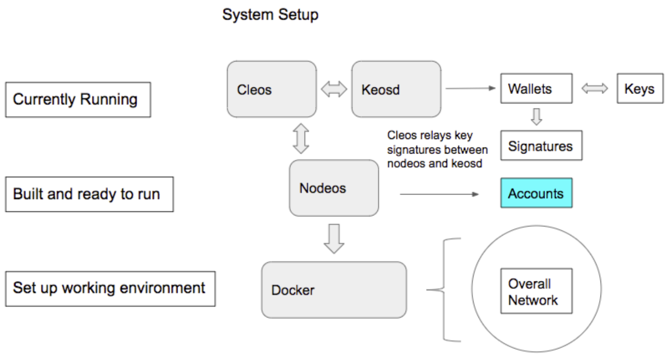


Fig. 2. How Accounts Fit Into the EOS System

foundational components and the relationship between state and contracts within the system.

2) Account Configuration: In the EOS architecture, accounts are presented as authorization structures that can define senders and receivers of actions. Hierarchical authorization can be granted through permissions management and contracts. Accounts can be granted permissions and configured to provide individual or group access for validating transactions [9]. They act as the base-unit data modules that allow sending and receiving of transactions in the system.

The creation of accounts coincides with key generation, where the keys created are associated with their corresponding wallets. This interaction is facilitated through the `Cleos` application. `Cleos`, `Keosd`, and `Nodeos` act in unison to publish the account within the network. Wallets are used to interact with the network through `Cleos` [4]. This workflow can be seen in Figure 2.

Wallets are represented as containers for public-private key pairs that are used to authorize operations performed in the system: `Keosd` manages wallets and access is facilitated through `Cleos` to allow user access to accounts [10].

3) Account Architecture and Setup: Accounts rely on permission-based structures that define roles within the system. `Nodeos` is the interface to the EOS network and both publishes accounts and manages their interactions within the network through `Cleos`. `Cleos` provides the ability for users to generate additional key pairs and interact with the RESTful interface.

`Cleos` is used to request `Nodeos` to create accounts and publish them into the network. As the `Cleos` tool is used to import private keys, the actions executed in the network will be signed by the keys affiliated with the account. When running multiple wallets, users must import the necessary keys into the wallet to create signatures for transactions [2]. Successful `wallet import` executions result in successful correspondence between the private key and public key within the system. As wallets contain keys, you can expect the wallet files to be stored in the `data-dir` or `/eosio-wallet` folder [4].

4) Account Functions: Accounts are used to execute contracts by sending structured actions to block producers. These actions are routed through the rest of the network in order to reach the destination accounts [4]. EOS defines contracts through actions and action handling scripts, which will be described in Section II.C. Accounts have two levels of permission: active and owner.

`Active`, the most generic account permission structure, is used for transferring funds, voting for producers, and committing high-level changes in the EOS system. It allows you to stake, transfer, vote, buy RAM and change the Active key. [2].

`Owner` accounts can execute any actions that will affect the authority of the account. It allows everything that the active permission does, but it can also change the owner of the account [2].

The `cleos create account` command will be executed from `Nodeos` to create an account with a user specified unique name. These accounts are required to stake EOS tokens in order to reserve RAM necessary to maintain the account [11]. The naming convention allows the characters [1-5a-z.], with 12 characters at most.

Hierarchical permissions can be created in this system, where permissions have different thresholds for the necessary signatures for transactions to be valid. This means that one account can permission another account to have some type of access. The newly permissioned account similarly can be given the right to allow other accounts to have more access, meaning that the permissions of an account can be chained together by multiple parties. The client facing side of the software architecture will log into `Cleos` to unlock wallets that manage how keys are granted to different account authorities [12]. To summarize the authority and permissions systems surrounding accounts:

- The permission management system is consistently checking for the right signature verification when it comes to transactions.
- Authority is distributed and compartmentalized across users in order to create hierarchical authorization structures.
- Control affiliated with accounts is weighted by the hierarchical structure that allows multi-user control of accounts.

EOS can define access to the necessary keys based on the relationship between accounts and permissions [4]. Through this configuration, multi-signature accounts and customized permission levels can be created through the `eosio.msg` contract and combinations of keys and accounts can be authorized to send actions to other accounts [2]. This hierarchical chain allows accounts to act on behalf of other accounts.

5) Analysis: An analysis of accounts in the EOS architecture indicates that accounts manage permissions and funds on the EOS network. Accounts in EOS follow a traditional configuration, similar to other cryptography based distributed systems. However, EOS accounts have another layer of abstraction beyond just a public/private key pair [4]. Accounts are congruent to user profiles, in which multiple key pairs and wallets, with varying permissions, can be tied to one account. To manage these, a user interfaces with `Nodeos` and `Keosd` through the command line tool, `Cleos`.

For comparison, Ethereum implements two types of accounts (contract accounts and externally owned accounts) in which the mapping of account addresses and account state are associated through cryptographic proofs that validate their state. Ethereum accounts are not actually user profiles, but instead represent public/private key pairs prevalent in blockchain architecture [13]. When comparing this to EOS, Ethereum accounts aren't reliant on constructs similar to user profiles, but instead integrate with the underlying cryptography.

EOS accounts rely on traditional trust mechanisms to validate actions. Should an account violate this trust, then the account can be frozen by block producers a concept that will be discussed in more detail in a later section. In spite of this mechanism, however, there is no guarantee that consensus will always result in the elimination of bad actors. The main difference between this setup and how accounts are created within Ethereum, is based on cryptographic proofs and validity. In Ethereum, a Patricia Merkle Tree [13] solidifies the state of accounts, whereas in EOS, state is instantiated within the database or table used to stream state (more on this will be explored in Section II.D).

Based on an analysis of the account model in the EOS system, it is clear that EOS accounts are not composed of the simple public/private key pairs implemented within other cryptocurrencies like Bitcoin or Ethereum: the EOS account model is more similar to a distributed database with traditional, permission based account systems. Accounts are utilized like user profiles rather than a simple public/private key pair with cryptographic validation [4]. Based on this assessment, the EOS network is not necessarily a blockchain-based cryptocurrency network, but rather a homogeneous distributed database network that allows different user accounts to communicate and interact through the distributed database network.

C. Transactions and Contracts

1) Overview: This section explores the relationship between smart contracts and the execution of transactions. Within blockchain environments, transactions act as computational events that dictate state transition. The way that transactions call upon existing contracts or initiate new ones is dependent upon the configuration of these contracts and how the system interface is designed. The way that transactions and contracts interact with a blockchain system defines certain levels of robustness that exist within the infrastructure.

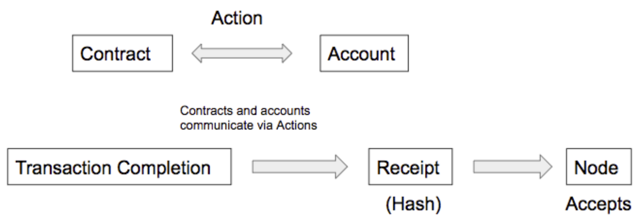


Fig. 3. EOS Actions

2) Contracts: Smart contracts in EOS are defined by a combination of actions and action handlers. Action handlers are designed to access different accounts through sending actions that can interact with the private databases affiliated with each account [4]. Block producers schedule transactions in order to optimize conflicts over memory and resource intensive actions within the network [14].

3) Web Assembly: Web Assembly (WASM) can be used as a compilation target for many high level languages such C++/C/Rust [15]. WASM is the instruction format that the EOS virtual machine understands, meaning that the interaction between the EOS library and WASM binary code happens through Web Assembly modules [16].

4) Components : **Contracts**: Programs that are registered in the network and executed on EOS nodes. Contracts can consist of action requests that are stored within the EOS database framework. Contract code is compiled into low level bytecode that can be executed by block producers [17].

Actions: These are the single atomic units of operation. Actions represent the communication interface between contracts and accounts. Most of the EOS network is built using actions that interact with each other and form the the underlying architecture [17].

Transactions: Transactions represent the execution of actions. Transactions can consist of individual actions or multiple actions [17]. The successful execution of transactions dictate the change of state in the EOS blockchain. The relationship between transactions and actions is shown in Figure 3.

EOS uses common contract functions to allow accounts to execute actions on behalf of users. EOS contracts are functionally similar to Ethereum-based ERC-20 contracts for the issuance of tokens [18].

5) Communications and Actions: EOS contracts can communicate and coordinate with one another, allowing dependencies between current and future actions. Communication between contracts is asynchronous. Further iterations of the resource distribution algorithm is designed to alleviate some of these concerns [2]. In EOS, contracts are software that are registered by nodes in the network. The process of sending an EOS contract is illustrated in Figure 4.

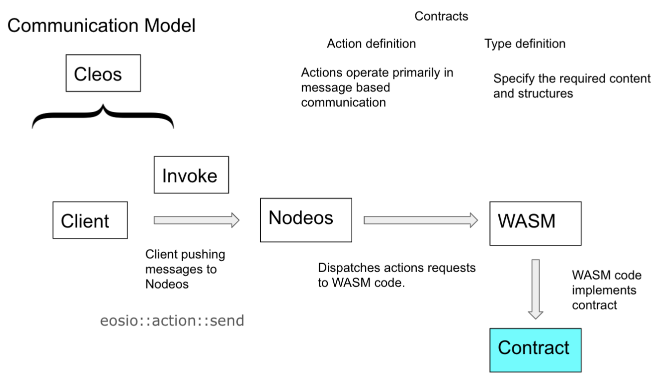

Communication Model
Contracts
Action definition
Type definition
Actions operate primarily in message based communication
Specify the required content and structures
Cleos
Invoke
Client
Nodeos
WASM
Client pushing messages to Nodeos
Dispatches actions requests to WASM code.
WASM code implements contract
eosio::action::send
Contract

Fig. 4. Contracts in the EOS System Architecture

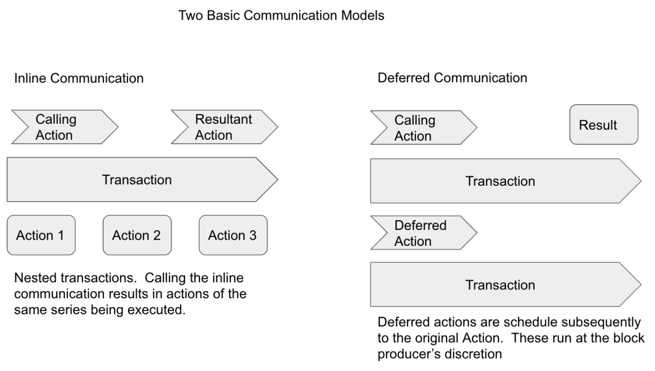

Two Basic Communication Models
Inline Communication
Calling Action
Resultant Action
Transaction
Action 1
Action 2
Action 3
Nested transactions. Calling the inline communication results in actions of the same series being executed.
Deferred Communication
Calling Action
Result
Transaction
Deferred Action
Transaction
Deferred actions are schedule subsequently to the original Action. These run at the block producer's discretion

Fig. 6. Two Basic EOS Communication Models

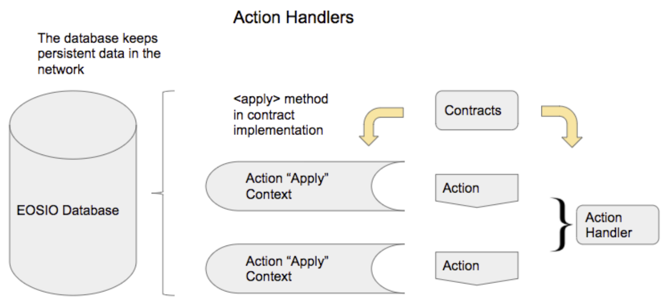


Fig. 5. Action Handlers Can Apply Multiple Contexts

Contracts and accounts are shown communicating via individual actions. When several actions are combined, they are packaged as transactions which execute the series of actions sequentially, as shown in Figure 5 [17]. Transactions are generated and a receipt is derived in the form of a hash. When nodes receive the transaction receipt/hash, there is high probability of the transaction being processed by the block producer, though it is not 100% confirmed yet [10].

Transactions can be constructed through two different types of communication models, as shown in Figure 6. These models are designed to provide granular control over actions that require immediate attention, as opposed to deferred actions that require more complex business logic [19].

The manner in which contracts are executed is important for the database model to execute necessary Database Management System (DBMS) functions. The context of actions reference three variables [4]:

- Receiver: the account currently executing processes
- Code: the account that authorized the contract
- Action: the ID of the currently running action

The action context also contains [4]:

- Current Transaction Data:
  - Transaction headers
  - Ordered vector of all original actions
  - Vector of context free actions
  - Prunable set of context free data
  - Full index of blobs
- Transaction Side Effects:
  - Changes to state in the EOS persistent storage
  - Notifications to the recipients of the current transaction
  - Inline action requests to send to a new receiver
  - Generation of new (deferred) transactions
  - Cancellation of existing (in-flight) deferred transactions (i.e. cancel already-submitted deferred transaction requests)

The working memory of an action is not shared with other actions in the same transaction as access is closed off based on permissioning across accounts. Variables are not available across action contexts and state is only passed among actions through persisting it in the EOS database [21]. Components in the system are compartmentalized for maximum efficiency in terms of processing.

6) Analysis: EOS uses a contract based model to build out the overall system. This model permeates to the lowest level of the protocol, as many of the system contracts, including the one defining the EOS token, follow this model. Contracts in EOS are more congruent with traditional lines of code that are designed to execute computational functions than the smart contract model defined and created with the Ethereum protocol. They are essentially plugins or modules within block producers, that users must interact with. Smart contracts in Ethereum are directly tied to state transitions, value transfers, and cryptographically determined algorithms that define an economic system. Gas in Ethereum is designed to measure the amount of computational effort expected to execute certain operations. The participants in Ethereum networks are also crypto-economically incentivized toward validating the network due to these fees generated from running contracts [22]. As state in Ethereum is cryptographically determined by instantiation in the form of Patricia Merkle Tree data structures, smart contracts are the computational logic components that result in state changes [23]. In Ethereum transactions, state transitions result in cryptographically validated changes to the underlying composition of the blockchain state as can be discerned through the changes to the Patricia Merkle Tree data structure. Conversely, transactions in EOS result mainly in changes to the underlying database, rather than cryptographically verified state changes to an underlying blockchain data structure like Ethereum.

As demonstrated by our previous analysis of the EOS system architecture, the communication model results in data being invoked through the client to eventually being processed by `Nodeos` [4]. All of these actions operate in an environment that lacks cryptographic validation of the contracts and transactions. Additionally there is no economic incentive mechanism that is enforced to facilitate proper execution in the system. This interaction offers services of a traditional computing environment with no cryptoeconomic system, as will be discussed in Section IV. Because EOS is built utilizing WASM in the context of normal database architecture, there are no substantial differences between EOS and what is expected in a traditional client/server architecture.

In EOS, when ABIs are generated to allow JSON-Binary conversion, the code that is processed relies on specifying the serialization/interface for interaction between human readable JSON and machine readable binary and does not exhibit the peer-to-peer characteristics of smart contracts on the blockchain. Alternatively, in Ethereum transactions, smart contract code is replicated by all nodes in the Ethereum Virtual Machine (EVM) to preserve the validity of the code and to ensure consistency, which contributes to the practical immutability of the contract and peer-to-peer execution [13]. How smart contracts are compiled in the EVM is vastly different from the way WASM is interpreted in EOS as WASM is used for generalized computing and EVM bytecode pertains to blockchain computing. This distinction points further to the dynamic that EOS does not necessarily implement a blockchain platform, though rather a generalized computing platform. In EOS, contracts are specifically modifiable and historical compositions of contracts can be altered. This model for contracts is effective for the command and control system architectures you would expect in a distributed database, not a blockchain [24], [2].

As detailed in Blockchain vs DLT: Part1 [23], the dynamics of client/server and database architectures are explored in terms of how they relate to Ethereum. Legacy Web 2.0 companies like Oracle and IBM have set the standard for client/server architectures and this model has been referenced within the design of distributed systems. A dynamic that we notice today in our analysis of software platforms is that distributed ledger systems and homogeneous distributed database systems like EOS are all still influenced heavily by the traditional client/server architecture. This is a traditionally accepted configuration, though it is important to realize that Ethereum and other cryptocurrency networks are designed for peer-to-peer configurations. In the Ethereum protocol, computation and validation is distributed throughout every node that participates in the system, so there is no centralized server architecture from which clients must request access. Nodes run their own implementation of the EVM and process the relevant code [23], a stark contrast to the EOS system where the computation and validation of contracts is primarily executed by the 21 block producers.

D. State Management and Blockchain

1) Overview: State facilitates representations of information and events in a computing environment. It defines the interactions that occur between different contracts and can have different contexts when exploring the concept as it pertains to blockchains vs traditional database systems. As described by Gavin Wood in the Yellow Paper [25], a blockchain is a cryptographically secure, transactional singleton machine with shared state. The structure of state in these environments defines aspects of the system's integrity of the infrastructure level platforms. Foundational layer technologies need proper state management in order to ensure high fidelity data structures upon which subordinate protocols can be built. The composition of the blockchain infrastructure is necessary in determining the degree of usability and stability of the underlying technology as it relates to state [2]. In order to establish a level of cohesion within a strong base layer protocol, a system must ensure the validity and maintenance of the system's state. As blockchain becomes a technology that is focused on value creation and systematic implementations of new forms of digital assets, resolute base layer components are needed to handled the demands of comprehensive environmental loads.

2) Resources: A key distinguishing factor among the various systems and networks in our ecosystem is how state and resources are managed within the computational framework. EOS state is highly integrated into the inner workings of the token model [4]. To allocate resources, token holders stake their tokens. This entitles them to the applicable proportion of the resources. There are three main resources utilized in the EOS distributed system architecture. **Bandwidth and Log Storage (Disk)**: Bandwidth can be broken down into instantaneous and long term bandwidth, which essentially acts as the Log Storage of all actions that take place in the network. This store of actions is downloaded by all nodes within the distributed computing environment. The log of actions is used to recreate the state of all applications run on the homogenous distributed database management system that comprises the bulk of the EOS architecture. Bandwidth is allocated on a fractional reserve basis, meaning that unused bandwidth can be rented or delegated to different accounts [2]. **Computation and Computational Backlog (CPU)**: Computation also has instantaneous usage as well as long term usage. Computational backlog (debt) is considered the calculation that is necessary to regenerate the state of Action Logs that are housed in the Log Storage. Computational backlog needs to be carefully managed as it can face limiting factors when the need for computation grows too quickly [2]. Similarly to bandwidth and log storage, computation is also managed through a fractional reserve basis. **State**

Storage (RAM): In the EOS architecture, state storage represents information that coincides with application logic. Block producers will publish their available capacity for bandwidth log storage, computational debt, and state storage. State storage availability is dependent upon the amount of tokens that an account stakes [2]. Unlike Bandwidth and computation, state storage cannot be rented or delegated to other accounts. Storage of application state requires the account to have tokens staked in the network.

3) Defining State: State is persisted across actions through EOSs DBMS infrastructure. This optimizes the facilitation of actions and transactions that are actively stored in the multi-index table. State persistence is necessary for the system to keep a record of transactions and data; modifications of application state require write access in this architecture and are conducted sequentially [19]. Computational debt used to recreate application state from the Log Storage, through validation logic, can be separated into three sections [26]:

- Validation that actions are internally consistent
- Validation that preconditions are met
- Modification of the application state

The read only aspects of this process can be conducted in parallel, while modification which requires write access must be conducted sequentially, when changes to the state are involved as expected in shared lock/exclusive lock systems [19], [27]. As actions from contracts operate within the action context, several services are provided to better facilitate the execution of the action:

- Actions are given access to working memory which allows the working state to exist [4].
- When new actions are executed, a new reserve of working memory is allocated because the context of actions are insulated from one another in that each action operates independently without a posteriori knowledge of any other.
- The persistent state of this network is allocated to these action contexts through the EOS database which stores and distributes this state information [26].

4) EOS Database: The model for object storage allows retrieval functionality which utilizes the multiple indices through the `eosio::multi_index` C++ class. This allows the contracts to read and modify the state history that exists in the EOS database. As with traditional databases, `eosio::multi_index` allows the index of tables to be user-defined functions over a range of columns and rows [28].

- The C++ interface provides a homogeneous container that is organized by multiple indices through keys derived from the objects.
- This structure is comparable to Boost multi-index containers, and conceptually can be considered tables in a traditional database setting [4].
- EOS multi-index DB also has iterators that are analogous to C++ database iterators.

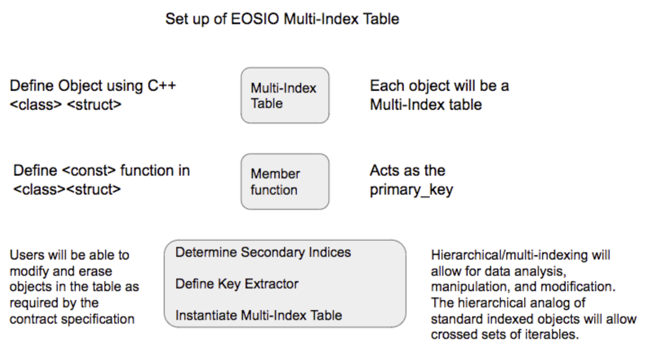

Set up of EOSIO Multi-Index Table
Define Object using C++ <class> <struct>
Multi-Index Table
Each object will be a Multi-Index table
Define <const> function in <class><struct>
Member function
Acts as the primary_key
Users will be able to modify and erase objects in the table as required by the contract specification
Determine Secondary Indices
Define Key Extractor
Instantiate Multi-Index Table
Hierarchical/multi-indexing will allow for data analysis, manipulation, and modification. The hierarchical analog of standard indexed objects will allow crossed sets of iterables.

Fig. 7. Overview of How EOS MultiIndex DB Works

5) Multi-Index Table : The RAM infrastructure of EOS utilizes multi-index tables to store state data and allows for create, read, update, and delete (CRUD) operations. The database provides an accessible method for storing data affiliated with smart contracts where actions can not access the contexts of other actions [29]. Two main components include:

- `Code` representing the contract account
- `Scope` representing the particular account that is responsible for the RAM fee and user of the contract; where contract is deployed

The setup of these multi-index tables is shown in Figure 7.

The multi-index tables of the EOS Database are used as the direct source of reference for the state affiliated with the network. As the schema of the table is defined, `eosio::multi_index` will be responsible for setting up the contract that instantiates the table [29]. `Nodeos` instances keep the database within memory by providing access through the REST API that reads the database. There are multiple methods of querying the database. On the client side, `Cleos` using the RPC API will have access to the database state. Access among accounts can also be authorized through the `require_auth` method provided by `eosio.cdt` which accepts an argument `account_name` and designates access authorization [5]. The `eosio.token` contract keeps track of balances of users in the database [4].

6) Analysis: As described in previous analysis, the EOS system architecture is not a blockchain, according to traditional definition, but rather a non-autonomous homogeneous distributed database system. As can be laid out within the specification of the EOS architecture, the system was created based on distributed database structures. It does not share the same characteristics which traditionally define a blockchain, but more closely resembles a DBMS, and more specifically functions as non-autonomous homogeneous distributed database. The EOS system is designed to replicate significant functionalities as can be expected from the finalized database. The system is designed to assign relational database properties to the boost multi-index table [28]. This stores data which is shared among participants in the network through a boost library call that streams data between each user. Its table is updated as actions are pushed through the system, though most transactions and movements of information are stored within the database structure of the system. State within EOS is dependent upon storing data inside a database where no cryptographic proofs or validation are used. What is considered state in the EOS system infrastructure depends primarily on how data is stored in the multi-index table. In EOS, there is no mempool for accessing pending transactions, as the multi-index table is not open source. Transactions are processed in a black box resulting in further layers of obfuscation. As far as cryptographic properties are concerned, there are loose properties affiliated with the hashing of transactions between each other, as well as an Incremental Merkle component, though it seems that most data is streamed between the database repositories within the system. The entire database can be rolled back indefinitely. In this system, consensus is not designed to validate transactions but rather to maximize the system's ability to process a higher degree of transactions. From this architectural design, it becomes further apparent that this system should not be characterized as a blockchain [4].

E. Execution Environment

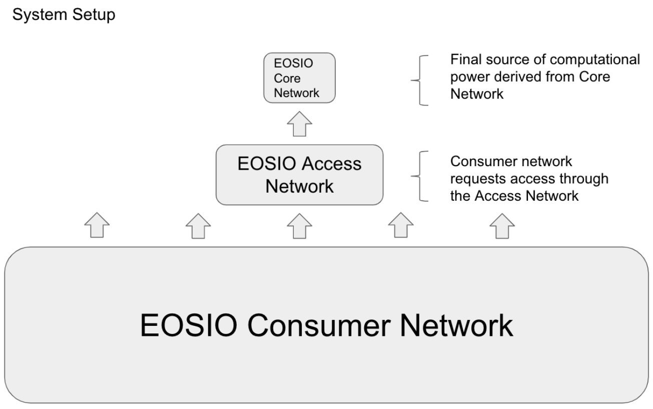
1) Overview: The execution environment and node set up within the network determines infrastructure level tendencies. The node setup will determine certain facets of decentralization, scalability, network robustness, and overall fault tolerance in real world environments. Controlling for differences in configuration, it is important to get an overview of how messages are propagated through the underlying system.

2) Setup: EOS is a full-scale network with a large amount of servers plus an ecosystem of users that aim to use the platform for computational resources and instantiation of applications within the hosted network environment [30]. The network appears to be demarcated between three main layers, as illustrated in 8:

- EOS Core Network: the innermost layer
- EOS Access Network: encapsulates the Core Network
- EOS Consumers: network that is accessed by a global community

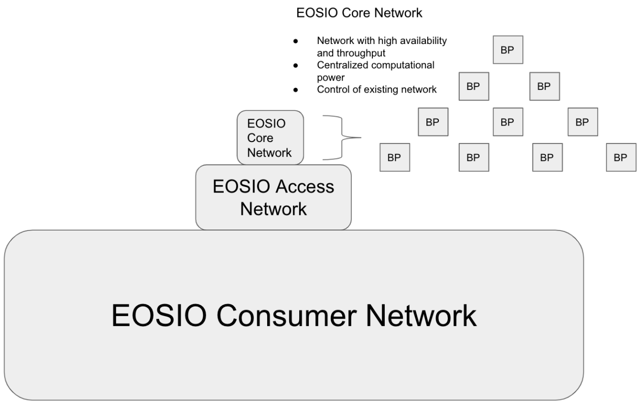
In this software architecture, one distinction to be made is that the various layers do allow movement between domains where there is no enforcement of boundaries between the stack [2]. This setup within the homogeneous distributed database allows for free sharing of information across the system towards allowing different users to gain access to the computation that is controlled within the server architectures of the block producers [30].

3) EOS Core Network: The EOS core network consists of block producers that are composed of industrial scale servers designed to provide distributed computation to all of the users in the mesh network, as shown in Figure 9. The 21



System Setup
EOSIO Core Network
Final source of computational power derived from Core Network
EOSIO Access Network
Consumer network requests access through the Access Network
EOSIO Consumer Network

Fig. 8. Network Access Layers of the EOS System



EOSIO Core Network
Network with high availability and throughput
Centralized computational power
Control of existing network
BP
EOSIO Core Network
EOSIO Access Network
EOSIO Consumer Network

Fig. 9. The Role of The Core Network in EOS

block producer nodes are designed to house superior computation in order for the receiving peers to obtain maximum computation power from the producers [30]. The architecture of the main network of servers provides the redundancy needed to allow for the activation of failsafe servers should the primary servers be unable to function. Servers are designed to be protected by powerful firewalls. A single block producer can be comprised of multiple industrial-grade servers with redundant disk space and supplemental capacity. The identities and locations of the block producers are kept anonymous for the safety of the network. The servers are designed to have high availability for network processing, while blocking out access from any non-affiliated third party [30]. The block producers use the exorbitant computational power they have available to synchronize execution for providing throughput and processing distributed through the homogeneous database architecture [30].

4) EOS Access Network: The EOS network was designed to provide horizontal scalability in a distributed network environment. This scalability is achieved by placing the bulk of the processing power within the EOS core network [2]. In order to ensure that the core network is able to maintain focus on providing higher computational processing abilities, the EOS access network filters requests from EOS consumers as

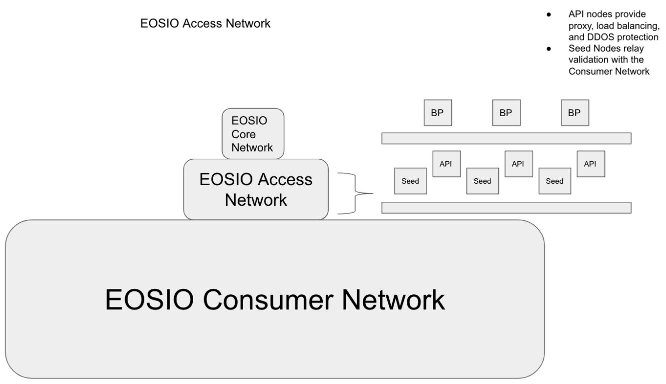

EOSIO Access Network
API nodes provide proxy, load balancing, and DDOS protection
Seed Nodes relay validation with the Consumer Network
BP
BP
BP
EOSIO Core Network
API
API
API
EOSIO Access Network
Seed
Seed
Seed
EOSIO Consumer Network

Fig. 10. The Role of the Access Network in EOS

shown in Figure 10. Instead of processing transactions, the access network layer is designed to alleviate the processing overhead of the core network [30].

The access network relies on several of the same security precautions and firewalls to prevent outside access into this network. Access network nodes can also someday become block producers in the core network if credibility is established and the nodes are successfully voted by the token holders [30]. The designation of primary roles depend on a set of plugins, while nodes are theoretically able to perform any action. Formal functions of the access network include: proxy services, load balancing, and DDoS protection. The access network contains two additional types of nodes: API nodes and seed nodes [30].

API nodes offload work from other nodes and operate behind a proxy or load balancer. They are designed to handle pre-processing transactions and to decrease the likelihood of transactions making their way to producing nodes. API nodes also process action requests, filter out bad transactions, and relay good transactions to a block producer or other nodes [30]. Each producer has one associated API node.

Seed nodes help connect to the network and track the blockchain. They are in charge of propagating blocks throughout the network, though not all seed nodes will validate the blocks [30]. By maintaining synchronization with the block producers, seed nodes may be able to become block producers if they function correctly and obtain enough votes [30]. Seed nodes only communicate using the EOS network protocol but do not process transactions, are not configured to run HTTP protocols, but cannot be accessed via `Cleos`. Block producers all have at least one associated seed node [30].

5) EOS Consumer Network: The consumer network consists of all of the groups who access EOS. Consumers generally gain access via `Cleos` or direct use of the EOS RPC API. Validating nodes assist by relaying information between block producers and consumers [30]. In order for the consumers of the network to access the computational resources of the system, they need to request access first via the access network, and then eventually aim to gain computational resources from the core network layer.

6) Analysis: Further information regarding the EOS architecture can be elucidated through understanding of the execution environment setup and how nodes are configured in the ecosystem. In order for consumers to be able to access the EOS computational resources in the network, they need to bypass several layers of control to eventually obtain access to the network [2].

Since the network is architected from a top-down perspective which limits decentralized participation in performing essential network functions, users must navigate through several layers of software hierarchy [2]. This is further illustrated in the relationship between the core and access networks.

This type of centralized configuration is optimal for top down command and control system infrastructures though may not necessarily be effective for decentralized systems that are designed to be base layer blockchain protocols. The EOS Core Network is comprised of non-transparent servers that do not share the source code that is being processed due to firewalls and insulation of the network. Given this structure, algorithmic or cryptographic proof implemented to validate that the block producers are actually running the same software or client presented within the publicly available EOS repository (this will be further explored in the Section III).

As mentioned in previous analysis, EOS architecture mirrors what would be expected in a client/server model like Amazon Web Services (AWS) where access to most of the computational and storage capabilities of the software are centrally stored by the resource providers [4]. In the case of EOS, the resource providers are the 21 block producers and their server architectures that are designed to provide computational resources to clients. EOS users do not share any resources and must request access via a servers consent prior to gaining any resources.

Alternatively, Ethereum reduces the distinction between the client/server interaction. Because nodes communicate through a peer-to-peer architecture, every node or client that is participating in Ethereum is able to contribute to the validation of the network as well as accessing the network. Ethereum nodes exhibit characteristics of both the client as well as the server in this network, showcasing a cornerstone of how this new decentralized model is approached in the Web 3.0 peer-to-peer space [23]. In Ethereum, any individual is able to personally set up an Ethereum node and participate in the network. There is no need to request access from middle layers in the network or request permissions from centralized servers. Setting up an Ethereum node will entitle an individual to participate in the network and validate transactions. In EOS, consumers are not entitled to this right, as they are required to request access from block producers. Based on these findings, it becomes clear that EOS is fundamentally similar to a centralized cloud computing architecture without the fundamental components of a blockchain or peer-to-peer network. EOS block producers are highly centralized and users can only access the network using block producers as intermediaries. Block producers are a single point of failure for the entire system.

F. Consensus

1) Overview: The final component of a decentralized system is how different nodes come to agreement on what has happened in the network. In permissionless networks, it is also important that there are mechanisms that prevent spam and Sybil attacks on the network. The EOS protocol uses a DPoS mechanism that implements a voting mechanism designed to help prevent actors from controlling multiple nodes in the network to their advantage [2]. The bulk of consensus comes from the voting and delegation in the network.

2) Multiple Stages: There are two stages to the DPoS algorithm. The first stage is staked voting, which is how the algorithm allows entities to assume block producer roles in the network for the purpose of processing transactions. In this voting scheme, token holders stake tokens to vote for potential block producers. The top 21 block producers with the most tokens voting for them become the block producers for the next epoch. The voting system is critical to EOS design, as it theoretically enables token holders to vote for block producers that provide lots of resources to the network and vote out block producers that act maliciously [2], [31].

Once the block producers are selected for the next epoch, the algorithm uses a traditional transaction processing mechanism between the block producers to agree upon transactions and contract execution. Scheduling allows each block producer to generate 6 blocks in a row at 0.5 second intervals [2]. Each epoch lasts approximately 126 seconds (enough for each block producer to generate 12 blocks apiece) and the votes in the system are then recalculated to identify the next group of block producers

3) Execution: When a block producer is generating the next block in their database, they should be executing and validating the transactions and contracts that clients in the network are sending them. The block producer processes the valid transaction and contracts and includes them in the storage layer while filtering out the invalid transactions and contracts. Once the data set has been processed and constructed, the block producer broadcasts the information to the other block producers in the network. Once a majority of 15 block producers have validated and signed the data set, the transactions are finalized [2].

If a block producer does not produce a set of transactions while active, the other block producers confirm an empty transaction set and move on to the next slot in order to ensure the productivity of the network [2].

4) Transactions as Proof of Stake: In EOS, each transaction refers to a block number which implements an additional layer of consensus. Block producers are not necessarily validating blocks, but are rather bundling transactions in a manner that would be expected in a data streaming service. Transactions are associated with previous blocks in the network and an Incremental Merkle is implemented in parallel to the multi-index table. The Incremental Merkle does not interact directly with state, but is rather used ephemerally as a data structure [2], [31].

5) Analysis: Because the bulk of consensus relies on the voting and delegation, any instance in which those actions can be corrupted completely negate the presence of Byzantine Fault Tolerance (BFT). DPoS is designed to stop groups of actors from controlling the network, though in practice, the voting system is primarily used to determine which block producers can control the network in its entirety . Due to token concentration, EOS token holders ensure that a cohort of block producers that are voted into the system are allowed to process transactions while unencumbered by non-block-producers. Meanwhile, once the block producers are selected, the algorithm operates similarly to other traditional DBMSs where transaction throughput is maximized for data processing.

If a block producer fails or acts in a malicious manner, the intent of the EOS protocol is for these block producers to be voted out. As discussed later in the report, sometimes the economics of this voting system can prevent this from happening. Based off the voting system and governance of the block producers, there are serious questions about the EOS systems ability to adequately deal with these bad actors while preventing centralization. Conceptually, it is impossible for EOS to implement Byzantine Fault Tolerance. A true BFT system would not be susceptible to cartels forming in the system. As will be described in further sections including the Performance Benchmarking section as well as the Economics section, cartels are easily formed in EOS, therefore negating any effort to claim BFT.

There are also problems with the transaction as proof of state system implemented in EOS. The network is susceptible to attacks in which the network can be flooded with false transactions that are associated with legitimate transactions from the past. The database structure of this system can be characterized as ephemeral, while concurrent changes can be rolled back indefinitely. The Incremental Merkle is susceptible to timing attacks as transactions are not necessarily processed sequentially though rather subjectively based on ease of processing (more of this will be explored in the Section III). Data structures in EOS are more continuous than they are discrete. Unlike proof of work systems which offer probabilistic finality, EOS does not necessarily offer any finality. As the entire database structure is based on Bitshares, one of the key defining characteristics is the ability to revert state and make changes that can date back to the genesis block.

In EOS, there is a lack of determinism or algorithmic enforcement of the consensus mechanism. Block producers subjectively come to consensus on the transactions that are

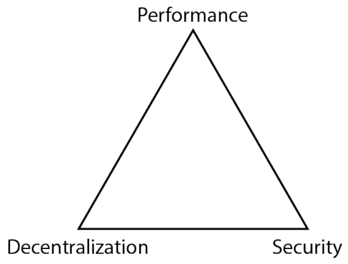


Fig. 11. Z. Cole's Triangle

received from other block producers without necessarily relying on any particular cryptographic proof in order to validate state. In blockchain technology, there is an effort for state to pass in discrete manners, though in systems that aim to implement parallel processing like EOS, there are a lot of problems in dealing with non-determinism along with deadlocks and race conditions. Block producers do not have direct visibility into transactions until they are already implemented within a block, meaning that block producers can only act retroactively regarding faulty/nefarious transactions, creating further doubt on the verification of consensus. EOS has very limited cryptographic proofs and is not algorithmically enforced, leading to the conclusion that BFT consensus is theoretically impossible in EOS and the network should not be characterized as having any form of BFT.

III. Performance

A. Introduction

When evaluating the architecture of a blockchain system, it is important to be aware of three particular metrics which characterize their design as illustrated in Fig. 11.

These three metrics are intrinsically correlated: one can only be optimized at the expense of another. Since they are generally asynchronous and distributed in nature, blockchain protocols tend to exhibit a lower level of transactional throughput in comparison to traditional client-server architectures. This is often the result of computationally exhaustive consensus algorithms and security mechanisms which are implemented in order to eliminate the need for trust when transmitting data within inherently adversarial environments. The ability to conduct trust-less peer-to-peer transactions, validated and ensured by cryptographic proofs, differentiates blockchain technology as a unique and innovative solution for privacy and censorship resistance within an increasingly interconnected global community. The aforementioned defines aspects of decentralization and security characterized by popular blockchain systems such as Bitcoin, Ethereum, or ZCash.

Determining the validity of a change in state is a security function. Increasing the speed of this process is a performance optimization. Performance optimizations are generally made at the cost of security since reducing computational overhead tends to present vulnerabilities within the overall security of the system itself. However, this does not imply that computationally exhaustive consensus algorithms are exclusive to security since game theory dynamics can compensate for this trade off when carefully designed and implemented. Since mechanism design can be a difficult undertaking, this loss can be compensated by limiting the publics participation or direct interaction with a blockchain networks core governance system, resulting in a higher degree of centralization.

The intent of these performance tests was to evaluate the trade-offs the EOS system makes in relation to the three metrics listed above, observe the practical limitations of its performance, and additionally cite an essential misunderstanding in the way the blockchain development community tends to identify and address performance bottlenecks.

B. Overview

The test series and system analysis was conducted on the Whiteblock blockchain testing platform which presents a deterministic framework offering such functions as the provisioning of multiple nodes, configuring the network conditions between these nodes, and automating their transactions and behaviors within a deterministic and controlled testing environment.

Since no standardized testing methodologies currently exist within the blockchain space, the following methodologies will adhere to the Whiteblock testing methodology, which presents a combination of performance, security, and concurrent testing models in order to accommodate for the dynamic, distributed, and asynchronous nature of blockchain systems.

In order to obtain a conclusive and objective understanding for the holistic functional performance capabilities of the EOS system, a real-time network was engineered within a laboratory environment. The specifications of this environment and its accompanying modules will be outlined in the following section. Subsequent sections describe the benchmarking methodology employed and the results of these experiments. We conclude with a presentation of the relevant data pertaining to the key performance metrics in our evaluation of the EOS system.

C. Scope

1) Software Version: The version of the EOS software that will be tested will be the latest version of the master branch on the EOS GitHub [31], or whichever version is presented as the primary active branch for production use, should there be a difference between the two.

2) Client Configuration: The config.ini file for each client will be configured based on the best practices recommended within the EOS Developers Portal [32], unless otherwise deemed sub-optimal, in which case the reasoning and supporting data will be provided.

Number of Servers	6
Model Name	PowerEdge R720
Vendor	Dell
CPUs	(2) Intel Xeon CPU E5-2667 v2 @ 3.30GHz
CPU Max MHz	4000.0000
NIC	82599ES 10-Gigabit SFI/SFP+ Network Connection
RAM	256GB DDR4

TABLE I
SYSTEM SPECIFICATIONS

3) Resources: In order to come to a conclusive and objective understanding for the holistic functional performance capabilities of the EOS system, a real-time network will be compiled and deployed within the Whiteblock framework. The specifications of this environment and its accompanying modules will be outlined in a later section.

D. Test Environment

1) Overview: The proposed testing initiatives will be conducted within a span of 6 rack-mounted chassis servers which have been engineered in order to most accurately reflect the computing environments provided by EOS block producers themselves. An example of an EOS block producers self-reported system specification can be found on the EOS New York website [33].

2) System Specifications: The testing environment was comprised of a network of 6 separate and high-powered machines which were engineered in order to most accurately reflect the computing environments provided by EOS block producers themselves. An example of an EOS block producers self-reported system specifications can be found on the EOS New York website .

Each machine hosted a number of virtual machines which varied between test scenarios. These virtual machines acted as client nodes within the EOS network. Each node ran the Ubuntu 18.04 LTS operating system. A single instance of the EOS client was run on each node which was was configured and managed through the Whiteblock platform to commit actions through the block producers. In each test, 21 block producers were configured in accordance with the client specifications presented within the EOS documentation. Each had a config.ini file and ran the Nodeos client. The source code was downloaded and compiled from the EOS GitHub repository [31].

3) Network Topology: The nodes were peered within a single chain, i.e., a network refers to a singleton genesis block. The nodes are disconnected from the outside world, and as such peered only with each other, however, each node operated within its own isolated network and was assigned its own IP address to ensure their logical separation from one another, thereby allowing the Whiteblock framework to configure and implement network conditions, such as latency and packet loss to emulate real-world performance with a high-degree of accuracy.

E. Testing Configuration

Configurations were distributed amongst the nodes while in a paused state, such that client initialization is not included in any benchmark.

1) Test Procedure Per test case:

1) Build new network
2) Create accounts
3) Configure block producer nodes
4) Configure client/user nodes
5) Unless otherwise specified, assume 21 block producers per test scenario
6) Vote for appointed 21 block producers
7) Queue appropriate number of transactions
8) Send transactions
9) Timeout 10 seconds
10) Output raw data to DB
11) Parse and average data after each test
12) Tear down network
13) Format data and push to appropriate repo

F. Performance Metrics and Data Points

1) Block Size: Measure the average memory space an individual block occupies; plot over time.

2) Block Time: Measure the average amount of time it takes to generate the next block in the blockchain; plot over time.

3) Transactional Throughput: Measure the amount of time it takes for a unit of value to transfer from the account of one individual holder to another.

a) Block Throughput: is calculated based on the number of transactions within a block divided by the amount of time it took to create that block.

b) Chain Throughput: is an overall throughput that is calculated based on a historical average of block throughput values.

These two metrics can vary drastically from one another based on particular variables and other network conditions. For the sake of simplicity, when referring to throughput, unless otherwise noted, well be referring to block throughput.

4) CPU Usage: Measure the average CPU usage; plot the CPU usage over time

5) Network Usage: Measure the overall network usage; plot the overall network usage over time

6) Transaction ID: The unique hash pertaining an individual transaction.

7) Transaction Size: Measure and log the total amount of memory space an individual transaction occupies, including block headers and any accompanying metadata. Sometimes referred to as transaction weight.

Variable tested	Case 1	Case 2	Case 3
Network Latency (ms)	0	50	100
Packet Loss (%)	0	0	0
Total Clients	24	24	24
Amount Staked	20k	20k	20k
Sender Accounts	24	24	24
Tx size per client sent (KB)	137	137	137
Tx per client (tx/s)	200	200	200
Total Tx submission rate (tx/s)	4000	4000	4000

TABLE II

TEST SERIES A: NETWORK LATENCY

8) Accounts: Maintain a log of activity pertaining to specific accounts associated with transactional activity. Within the EOS system, an account is defined as a human readable string which is 12 characters in length. Each account refers to a users public and private keypair. Keypairs are generated using sekp256k1 elliptical curve parameters borrowed from Bitcoin.

a) Sender Account: refers to the account from which a transaction was sent.

b) Receiving Account: refers to the account that received a transaction.

9) Send Time: Maintain a log consisting of timestamps for each time a transaction was broadcast to the network.

10) Received Time: Maintain a log consisting of timestamps for when a sent transaction was available within the receiving account.

11) Total Time: Maintain a log for the total amount of time a particular test case or series was run. The first action within the test environment for each new test case or series marks the start time and the final action marks the end time.

12) Block Height: Maintain a log for the total amount blocks within a chain for a particular test series or test case.

13) Successful Transactions: Another important metric to note is that although transactional throughput remains fairly consistent across tests, the number of successful transactions decreased exponentially.

G. Performance Tests

The performance test of this investigation was conducted across a series of 6 different metrics.

Series A: Network Latency
Series B: Number of Transactions Per Client
Series C: Size of Transactions Per Client
Series D: Packet Loss
Series E: Number of Block Producers
Series F: Higher Numbers of Clients

Three test cases were involved in each series of the testing. **Table II-VII** indicate the topology setup, variables tested, along with the metrics affiliated with each test series.

Variable tested	Case 1	Case 2	Case 3
Network Latency (ms)	0	50	100
Packet Loss (%)	0	0	0
Total Clients	24	24	24
Amount Staked	20k	20k	20k
Sender Accounts	24	24	24
Tx size per client sent (KB)	137	137	137
Tx per client (tx/s)	200	200	200
Total Tx submission rate (tx/s)	8000	16000	20000

TABLE III

TEST SERIES B: NUMBER OF TRANSACTIONS PER CLIENT

Variable tested	Case 1	Case 2	Case 3
Network Latency (ms)	0	50	100
Packet Loss (%)	0	0	0
Total Clients	24	24	24
Amount Staked	20k	20k	20k
Sender Accounts	24	24	24
Tx size per client sent (KB)	259	344	429
Tx per client (tx/s)	200	200	200
Total Tx submission rate (tx/s)	4000	4000	4000

TABLE IV

TEST SERIES C: SIZE OF TRANSACTIONS PER CLIENT

Variable tested	Case 1	Case 2	Case 3
Network Latency (ms)	0	50	100
Packet Loss (%)	0.01	0.1	1
Total Clients	24	24	24
Amount Staked	20k	20k	20k
Sender Accounts	24	24	24
Tx size per client sent (KB)	137	137	137
Tx per client (tx/s)	200	200	200
Total Tx submission rate (tx/s)	4000	4000	4000

TABLE V

TEST SERIES D: PACKET LOSS

Variable tested	Case 1	Case 2	Case 3
Network Latency (ms)	0	50	100
Packet Loss (%)	0	0	0
Total Block Producers	3	5	7
Total Clients	24	24	24
Amount Staked	20k	20k	20k
Sender Accounts	24	24	24
Tx size per client sent (KB)	137	137	137
Tx per client (tx/s)	200	200	200
Total Tx submission rate (tx/s)	4000	4000	4000

TABLE VI

TEST SERIES E: NUMBER OF BLOCK PRODUCERS

Variable tested	Case 1	Case 2	Case 3
Network Latency (ms)	0	50	100
Packet Loss (%)	0	0	0
Total Clients	100	200	300
Amount Staked	20k	20k	20k
Sender Accounts	24	24	24
Tx size per client sent (KB)	137	137	137
Tx per client (tx/s)	200	200	200
Total Tx submission rate (tx/s)	4000	4000	4000

TABLE VII

TEST SERIES F: HIGHER NUMBER OF CLIENTS

H. Consensus Tests

1) Voting: Similar to BitShares, consensus within the EOS system is established via delegated Proof of Stake (dPoS), which is entirely dependent on community participation in order to effectively vote, or delegate, which nodes will be block producers.

This voting mechanism assumes that a large portion of the community will actually participate in this voting process, however, a low percentage of actual token owners seem to actually partake in this process, meaning that those who do vote are more than likely aligned with a particular block producer. Since there is no way to algorithmically prevent the formation of cartels, we can assume that the majority of voters are only doing so in order to ensure their own self-interest. This is an intangible data point that cant be deterministically measured through our testing initiatives, however, we can test the extent of voting manipulation which can be allowed within the network. Since users can vote through proxies, what is the extent of control these proxies have over the voting process? We can test this by attempting to deliberately withhold votes from the network rather than broadcasting them or pushing them to the smart contract responsible for voting.

When a user votes for a block producer, this vote is broadcast to the network as a transaction. If the account is blacklisted/frozen, the vote will not go through. This shows that the block producers will be able to keep their positions if they blacklist/freeze the right accounts. This might need some collusion as the blacklist will need to be shared among the majority of the block producers.

Block producers are making a substantial amount of EOS ($10000+) daily. This means that these block producers can simply create new accounts and/or use proxies and reallocate their EOS to those accounts to vote for themselves or other block producers with which they may choose to align forming something of a cartel of block producers. This way, these block producers can stay a block producer, even if it is as a different node/account, and their allies remain in control of the network.

2) Sybil Attack: Block producers have the ability to blacklist accounts. Should a malicious actor (or group of malicious actors) attempt to perform a Denial-of-Service attack on the network, the block producers can simply limit the accessibility of the account via blacklisting, however, this method of deterring Sybil-like behavior can only be enforced once a block producer has identified a malicious actor, consensus has been achieved in order to blacklist the account in question, and it has been manually blacklisted. Although it costs a minimal amount of EOS to create an account, the black listing process is slower than the rate at which one can create new accounts and submit transactions, rendering this process ineffective in reducing the occurrence of such attacks.

Another mechanism to prevent such events is addressed through the use of API nodes which are implemented on the behalf of the block producer to provide functionality similar to load balancers. Since the creation of new accounts is inconsequential and transactions incur no associated fees, one can assume the presence of these load balancers/proxies to be a minor inconvenience to whomever has the technical capacity to attack the EOS network. This hypothesis can be tested by identifying the upper boundaries of transaction capacity within the test environment and then intentionally exceeding these boundaries for a prolonged period of time in order to observe the effects on the network.

The EOS blacklisting mechanism requires all 21 block producers to share the same blacklist, which is not broadcast between all block producers. Each block producers blacklist is stored locally, yet the producers would need to share their blacklists with one another to some degree. The limits of this functionality can be practically tested by evaluating whether or not an account, once blacklisted by a specific block producer, can simply wait for the next round of block production by a different producer in order to ensure their transactions are pushed via another producer. This also reveals the lack of broadcasting/synchronization of nodes (particularly block producers in this case), which shows that the EOS network is not relying on peer-to-peer communication and there is a huge disconnect between block producers.

3) Client Software: In his 2004 research describing Reusable Proofs of Work [34], Hal Finney also presented the concept of the transparent server in which distributed network servers provide a degree of proof that the programs they are running are indeed generated by the code they present to the public. To bring this concept into context, block producers within the EOS network offer only the degree of transparency which they decide to offer. There is no guarantee that the block producers are running the correct client. Since most of the security mechanisms within the EOS system rely on the intentions of the block producers themselves, there is no algorithmic enforcement or available method to validate the EOS software that the block producers are running.

While more traditional blockchain systems evoke some degree of game theory to avoid the need for trust within inherently adversarial environments, actors within the EOS network are required to simply trust that the block producers are acting in their best interest without any of the cryptographic or algorithmic mechanisms that distinguish blockchain systems from traditional financial constructs. Proponents for dPoS consensus would claim that these sacrifices of security are made in order to enhance the overall performance of the network, but these data points can be objectively evaluated within this test series.

To what degree can the source code be changed? We tested this by making our own changes to the source code and analyzing the output performance data.

4) TPOS Vulnerability: TPoS is found to have some inherent vulnerabilities and the reasons for these are as follows:

- Transactions only require the hash of the most recent block header to prevent transaction replay on different chains and to signal the network that a user and their stake is on a particular fork. There is no additional validation needed for transactions to be added to a block.
- Transactions are referenced by a multi-index table, but this does not ensure that the block producers have not tampered with the state (see section above **Fault Tolerance**).

I. Results

1) Series A: The purpose of test series A is to assess the network under varying network latency conditions and to set up a base case to compare results with the other test series. Round trip time (RTT) within the network is varied between 0 ms, 50 ms and 100 ms. As shown in Table II, high levels of TPS are sustained at 0 ms round trip latency. When 50 ms of RTT is added into the system, performance immediately falls below 50 TPS indicating that latency has a significant effect on throughput. Once RTT is increased to 100 ms, performance drops even further.

2) Series B: Series B aims to vary the number of transactions sent per client. As can be seen through the data, the number of transactions sent per client has limited effect on the overall performance in terms of TPS. Though further studies can be done on the transactions success rate as it pertains to the number of transactions sent per client.

3) Series C: Series C indicates that transaction size has an apparent effect on overall throughput. Empty transactions are around 137 btyes, and throughput is showcased in Series A test trial 1. As transaction size increases, the performance drops.

4) Series D: The data indicates that packet loss has a direct effect on performance. Under real world packet loss conditions such as 0.001% packet loss or 0.01% packet loss, TPS suffers tremendously to the point where performance is close to on par with Ethereum levels of TPS.

5) Series E: Series E varies the number of block producers in the network. The results show that here is a positive correlation between the number of producers and throughput. However, further studies are necessary to gain a better insight into this trend.

6) Series F: The number of clients in the system was varied from 100, 200, to 300 clients. As can be discerned from the data, the effect of increasing the number of clients in the network did not have significantly pronounced effects. This test can be further extrapolated to larger amounts of clients to test the overall resilience of the network to real world conditions.

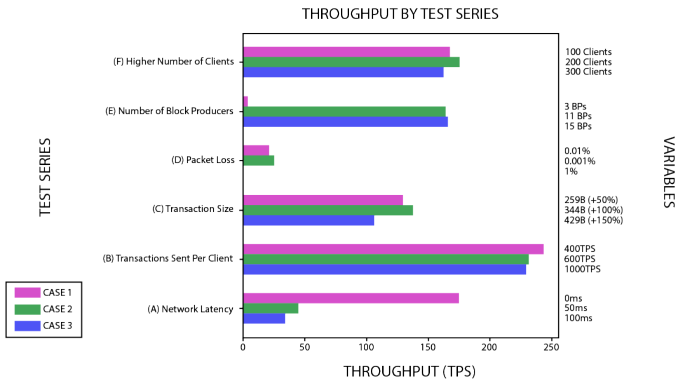


Fig. 12. Throughput By Test Case

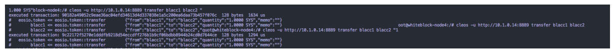
Series	Ratio	Percentage
A1	0.95972	95.97
A2	0.60435	60.44
A3	0.16778	16.78
B1	0.95972	95.97
B2	0.60435	60.44
B3	0.47781	47.78
C1	0.96396	96.40
C2	0.95985	95.99
C3	0.96023	96.02
D1	0.16778125	16.78
D2	0.16778125	16.78
D3	0.0	0.0
E1	0.001645	0.16
E2	0.458333	45.83
E3	0.458333	45.83
F1	0.29111	29.11
F2	0.14535	14.53
F3	0.09688	9.69

TABLE VIII
SUCCESSFULLY PROCESSED TRANSACTIONS

J. Analysis

The performance metrics observed when factors such as real world network latency and packet loss are implemented indicate that TPS in EOS approaches levels comparable to Ethereum. Concurrently, increases in transaction size also result in lower transaction throughput. This leads to the prediction that the high levels of transaction throughput that are showcased in marketing documents were achieved under ideal network conditions, and not real world network conditions. Additionally it should be noted that EOS is capable of selectively choosing transactions to be processed in the system as opposed to using predetermined algorithms. This can lead to the network picking out smaller/easier transactions to process in order to maximize transaction throughput.

When assessing the success rate of processed transactions, it is apparent that the network does not alleviate congestion, though rather circumvents it. As transaction success rate can drop below 1% in certain trials, it can be assumed that the unprocessed transactions are kept in the heap. The fate of these transactions are currently unknown, as ignoring transactions in the heap may be another mechanism used to



```
1.000 SYS"block-node4:/# cleos -u http://10.1.0.14:8889 transfer blacc1 blacc2 "
executed transaction: 90182a49052c9eae36ac04efd34613d4d337038e1a5c200ea6daa73b457f076c  128 bytes  1634 us
#   eosio.token <= eosio.token::transfer        {"from":"blacc1","to":"blacc2","quantity":"1.0000 SYS","memo":""}
#        blacc1 <= eosio.token::transfer        {"from":"blacc1","to":"blacc2","quantity":"1.0000 SYS","memo":""}                                                 oot@whiteblock-node4:/# cleos -u http://10.1.0.14:8889 transfer blacc1 blacc2
#        blacc2 <= eosio.token::transfer        {"from":"blacc1","to":"blacc2","quot@whiteblock-node4:/# cleos -u http://10.1.0.14:8889 transfer blacc1 blacc2 "1
executed transaction: 9c22172f5270e1dddfb9218d54eccdff276b1b9cf06bdbb8944b24cd0d7644ce  128 bytes  1294 us
#   eosio.token <= eosio.token::transfer        {"from":"blacc1","to":"blacc2","quantity":"1.0000 SYS","memo":""}
#        blacc1 <= eosio.token::transfer        {"from":"blacc1","to":"blacc2","quantity":"1.0000 SYS","memo":""}
```

Fig. 13. Transaction Sent from blacc2 to blacc3

increase perceived transaction throughput

K. Manual Test

1) Sybil Attack Test:

a) Overview:

- Blacklist an account by majority of the block producers
- Send multiple transactions from the blacklisted account
- Check if these transactions will be successfully processed or rejected
- Observe which block producer has included blacklisted account transactions into a block

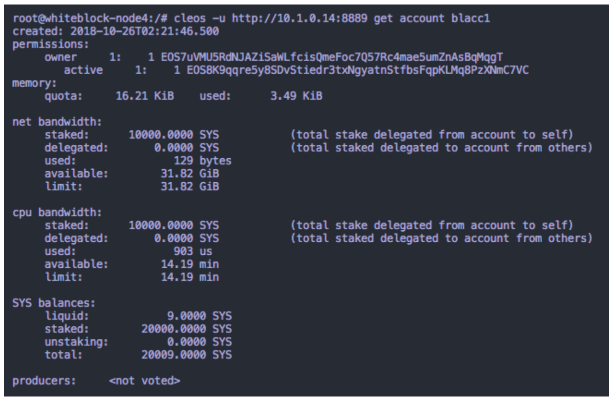
b) Steps:

- Create 3 BPs nodes
- Create 3 client nodes
- 2 block producers blacklist all clients
  - Add blacklist to config.ini
    - ”actor-blacklist = blacc1”,
    - ”actor-blacklist = blacc2”,
    - ”actor-blacklist = blacc3”,
  - Repeat for BP nodes 1 & 2
- Kill nodeos
- Reinitiate nodeos using updated config.ini file
- Have blacklisted account send transactions
- check block logs to see if any of the transactions have been included in the blocks

c) Results:
In this system, three block producers were created in representation of the network. The set up consisted of a genesis block producer for setting up the network along with three block producers; EOSIO, producer 2, producer 3, and producer 4. Each account in the network; blacc1, blacc2, and blacc3 were allocated 10 SYS each. In this network, block producer 1 and producer 2 have blacklisted accounts, blacc1, blacc2, and blacc3, while the producer 3 did not update their blacklist. When continual transactions are sent into the system, block producers 1 & 2 stop the transactions from occurring because of their updated blacklist though a transaction successfully gets executed when processed by the third block producer as described in Fig. 13.

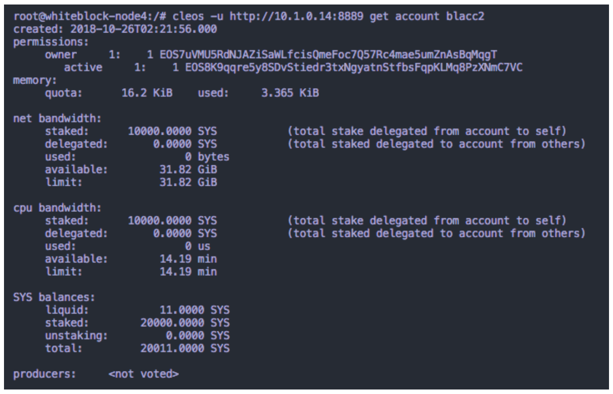
As indicated in Fig. 14 and Fig. 15, account blacc1 has 9 SYS while blacc2 has 11 SYS, indicating that a transaction was successfully processed as each account originally started with 10 SYS.

d) Analysis:
The implications of this successful transaction are that the blacklist does not have a coordinated consensus mechanism to algorithmically enforce its rules. The black list is merely a registry that requires trusted block producers to manage and



```
root@whiteblock-node4:/# cleos -u http://10.1.0.14:8889 get account blacc1
created: 2018-10-26T02:21:46.500
permissions:
     owner     1:    1 EOS7uVMU5RdNJAZiSaWLfcisQmeFoc7Q57Rc4mae5umZnAsBqMqgT
        active     1:    1 EOS8K9qqre5y8SDvStiedr3txNgyatnStfbsFqpKLMq8PzXNmC7VC
memory:
     quota:     16.21 KiB    used:      3.49 KiB

net bandwidth:
     staked:      10000.0000 SYS           (total stake delegated from account to self)
     delegated:       0.0000 SYS           (total staked delegated to account from others)
     used:               129 bytes
     available:        31.82 GiB
     limit:            31.82 GiB

cpu bandwidth:
     staked:      10000.0000 SYS           (total stake delegated from account to self)
     delegated:       0.0000 SYS           (total staked delegated to account from others)
     used:               903 us
     available:        14.19 min
     limit:            14.19 min

SYS balances:
     liquid:           9.0000 SYS
     staked:       20000.0000 SYS
     unstaking:        0.0000 SYS
     total:        20009.0000 SYS

producers:     <not voted>
```

Fig. 14. Balance of Account blacc1



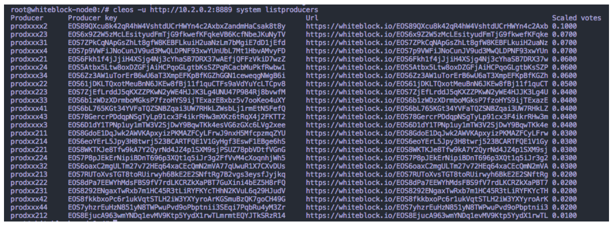
```
root@whiteblock-node4:/# cleos -u http://10.1.0.14:8889 get account blacc2
created: 2018-10-26T02:21:56.000
permissions:
     owner     1:    1 EOS7uVMU5RdNJAZiSaWLfcisQmeFoc7Q57Rc4mae5umZnAsBqMqgT
        active     1:    1 EOS8K9qqre5y8SDvStiedr3txNgyatnStfbsFqpKLMq8PzXNmC7VC
memory:
     quota:      16.2 KiB    used:     3.365 KiB

net bandwidth:
     staked:      10000.0000 SYS           (total stake delegated from account to self)
     delegated:       0.0000 SYS           (total staked delegated to account from others)
     used:                 0 bytes
     available:        31.82 GiB
     limit:            31.82 GiB

cpu bandwidth:
     staked:      10000.0000 SYS           (total stake delegated from account to self)
     delegated:       0.0000 SYS           (total staked delegated to account from others)
     used:                 0 us
     available:        14.19 min
     limit:            14.19 min

SYS balances:
     liquid:          11.0000 SYS
     staked:       20000.0000 SYS
     unstaking:        0.0000 SYS
     total:        20011.0000 SYS

producers:     <not voted>
```

Fig. 15. Balance of Account blacc2

implement. This further implies that there is not a true consensus algorithm in a BFT context as the system is constantly susceptible to spam and DDoS attacks. This is in fact a large vulnerability in the system as fraudulent users are essentially able to create malicious accounts much faster than the block producers are able to come to consensus on the content of their blacklist. Block producers do not actually process transactions based on any consensus algorithm, though rather process transactions in a mechanical fashion as there is no formal verification of the validity of transactions.

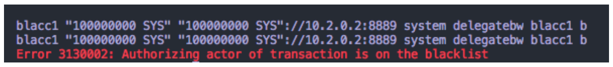
Malicious actors are able to send transactions that are fraudulent as long as they act before block producers can come to a social agreement to blacklist their account and unanimously update their blacklist. It is important to note that in order to update the blacklist, it is necessary to shut down <nodeos> and then restart the system while re-entering the network with the updated <config.ini> file. A skilled adversary is able to create multiple accounts at minimal costs, much faster than all the block producers coming to social agreement on a blacklist, update their config.ini file, and reinitiating their nodeos instance. Within the timeframe necessary to conduct that process, malicious actors can already execute multiple subsequent spam attacks, bypassing any blacklist.

2) Voting:



```
root@whiteblock-node0:/# cleos -u http://10.2.0.2:8889 system listproducers
Producer      Producer key                                           Url                                                    Scaled votes
prodxxxx2     EOS89QXcu8k42qR4hW4VshtdUCrHWYn4c2AxbxZandmHaCsak8t8y    https://whiteblock.io/EOS89QXcu8k42qR4hW4VshtdUCrHWYn4c2Axb 0.1000
prodxxx23     EOS6x9Z2W5zMcLEsityudFmTjG9fkwefKFqkeVB6KcfNbeJKuNyTV    https://whiteblock.io/EOS6x9Z2W5zMcLEsityudFmTjG9fkwefKFqke 0.0700
prodxxx31     EOS7ZPkCqNApGsZhLt8gfW8KEBFLkuiH2uaNzLm7bMgiE7dD1jEfd    https://whiteblock.io/EOS7ZPkCqNApGsZhLt8gfW8KEBFLkuiH2uaNz 0.0700
prodxxxx4     EOS7p9VWFiJNoCunJV9ud3MwQLDPNF93xwYUnUbL7Mt1HbvAMvyFD    https://whiteblock.io/EOS7p9VWFiJNoCunJV9ud3MwQLDPNF93xwYUn 0.0700
prodxxx21     EOS6Fkh1f4jJjiH4XSjg4Nj3cYhaSB7DRX37wAEfjQFFzVkiD7wzZ    https://whiteblock.io/EOS6Fkh1f4jJjiH4XSjg4Nj3cYhaSB7DRX37w 0.0600
prodxxx24     EOS5Atbx5Ltw8oxDZGFjAiHCPqoGLgtbKsSZPqRCacbMuPkfRwbw1    https://whiteblock.io/EOS5Atbx5Ltw8oxDZGFjAiHCPqoGLgtbKsSZP 0.0600
prodxxx34     EOS6Zz3AW1uTorErB6wU6aT3XmpEFKpBfKGZhGGN1ceweqgNWgB6i    https://whiteblock.io/EOS6Zz3AW1uTorErB6wU6aT3XmpEFKpBfKGZh 0.0600
prodxxx22     EOS61jDKLTQxotMeuBnN6JKEw8fBj11f1quCTFs9aVdYuYcLTCpv8    https://whiteblock.io/EOS61jDKLTQxotMeuBnN6JKEw8fBj11f1quCT 0.0500
prodxx223     EOS7ZjEfLrddJ5qKXZZPKwN2yWE4HJJK3Lg4UNU47P984Rj8bvwfM    https://whiteblock.io/EOS7ZjEfLrddJ5qKXZZPKwN2yWE4HJJK3Lg4U 0.0400
prodxxx33     EOS6b1zWDzXDrmboMGksP7fzoHYS9ijTExazEBxbz5v7ooKeo4uXY    https://whiteblock.io/EOS6b1zWDzXDrmboMGksP7fzoHYS9ijTExazE 0.0400
prodxxx41     EOS6bL765KGt34YVFaTQZSNBZqai3UW7RHkLZWsbLj1rmEtN5FefQ    https://whiteblock.io/EOS6bL765KGt34YVFaTQZSNBZqai3UW7RHkLZ 0.0400
prodxxx43     EOS78GercrPDdqpNSgTyLp91cx3F4ikrRHw3mXKz6tRqX4j2FKTT2    https://whiteblock.io/EOS78GercrPDdqpNSgTyLp91cx3F4ikrRHw3m 0.0400
prodxxxx3     EOS6D1dY1TPNp1uy1mTW3V2SjDwY9BqwTKk4esVG6zGXc6LVg2xee    https://whiteblock.io/EOS6D1dY1TPNp1uy1mTW3V2SjDwY9BqwTKk4e 0.0400
prodxx211     EOS8GdoE1DqJwk2AWVKApxyizPKMAZFCyLFrwJ9nxH5MfcpzmqZYU    https://whiteblock.io/EOS8GdoE1DqJwk2AWVKApxyizPKMAZFCyLFrw 0.0300
prodxx214     EOS6eoYErL5Jpy3H8twrj523BCARTFQE1V1GyHgf3EswF1EBge6hS    https://whiteblock.io/EOS6eoYErL5Jpy3H8twrj523BCARTFQE1V1Gy 0.0300
prodxx221     EOS8WKTKJe8Tfw9kA7Y2QyrNd4JZ4p1SXM9sjPSUZ78pbVDtfVGnG    https://whiteblock.io/EOS8WKTKJe8Tfw9kA7Y2QyrNd4JZ4p1SXM9sj 0.0300
prodxx224     EOS7P8pJEkErNipiBDnT696p3XQt1q5iJr3g2FfVvM4cXoqnhjWh5    https://whiteblock.io/EOS7P8pJEkErNipiBDnT696p3XQt1q5iJr3g2 0.0300
prodxxx32     EOS6oaxC2mgULTm27v72HEq64xaCEcQmN2mVA77qUwuR1X7CXvDUs    https://whiteblock.io/EOS6oaxC2mgULTm27v72HEq64xaCEcQmN2mVA 0.0300
prodxx213     EOS7RUToXvsTGT8toRUirwyh6BkE2E2SNftRg7B2vgs3eysfJyjkq    https://whiteblock.io/EOS7RUToXvsTGT8toRUirwyh6BkE2E2SNftRg 0.0200
prodxx222     EOS8dPa7EEWYhMdsFBS9fV7rdLKCRZkXaPBT7GuXini4bEZ5H8rFQ    https://whiteblock.io/EOS8dPa7EEWYhMdsFBS9fV7rdLKCRZkXaPBT7 0.0200
prodxx231     EOS8292ENgaxTwRxb7m1HC45R3tLiRYFKYcTHhN2KVuL6q29HJudV    https://whiteblock.io/EOS8292ENgaxTwRxb7m1HC45R3tLiRYFKYcTH 0.0200
prodxxx42     EOS8fkkbxoPc6r1ukVqtSTLH2iW3YXYyroArKGSmuBzQK7goCH49G    https://whiteblock.io/EOS8fkkbxoPc6r1ukVqtSTLH2iW3YXYyroArK 0.0200
prodxxx44     EOS7yhzrEuHzN851yN8TWPwuPvd9oPbptnii3SEqi7PqbRu4yM3Zr    https://whiteblock.io/EOS7yhzrEuHzN851yN8TWPwuPvd9oPbptnii3 0.0200
prodxx212     EOS8EjucA963wmYNDq1evMV9Ktp5YydX1rwTLmrmtEQYJTkSRzR14    https://whiteblock.io/EOS8EjucA963wmYNDq1evMV9Ktp5YydX1rwTL 0.0100
```

Fig. 16. Full List of Block Producers



```
blacc1 "100000000 SYS" "100000000 SYS"://10.2.0.2:8889 system delegatebw blacc1 b
blacc1 "100000000 SYS" "100000000 SYS"://10.2.0.2:8889 system delegatebw blacc1 b
Error 3130002: Authorizing actor of transaction is on the blacklist
```

Fig. 17. Failed Transaction from Client Due to Blacklisting

a) Overview:

- Every block producer will blacklist all voters
- Client nodes vote for different BPs
- Investigate if BP will keep their positions

b) Steps:

- Create 22 BP nodes
- Create 1 client node
- Give client node majority of the networks SYS
- BP1-21 will blacklist the client node
  - Add blacklist to config.ini
    - ”actor-blacklist = blacc1”,
- Kill nodeos
- Reinitiate nodeos using updated config.ini file
- Client node will attempt to vote for BP22
- Check if BP22 has become one of the 21 BPs

c) Results:

Fig. 16 represents votes that all of the block producers have. In this test, one account was manually given the majority of the SYS in the network. SYS was pulled directly from EOSIO and sent directly to account blacc1. A coordinated effort was executed where block producers would blacklist this account.

As Fig. 17 shows, the account and resulting transactions are now on the coordinated black list. shown in the same figure, the account blacc1 is attempting to delegate bandwidth to their account but they are not allowed to. This implies that the blacklisted account: blacc1 is not able to delegate anymore bandwidth, vote for block producers, or send transactions. The account is now completely frozen.

d) Analysis:

This type of attack can be executed by block producers simply accessing the explorer to discern the EOS holders with the largest voting powers and colluding to blacklist the accounts that threaten the authority of their existence as block producers. This type of vulnerability in the system is important to distinguish as there is no algorithmic rule preventing block producers from behaving in this manner. There is no proper protocol that is setup to prevent block producers from colluding to maintain their role as block producers. This further proves the high level of centralization that exists in the EOS network and the tremendous power these block producers possess.

3) Source Code & Client Software:

a) Overview:

- Make changes to the source code for one of the BPs
- Rebuild the network
- See if the BP and clients can communicate and participate in the network

b) Steps:

- make 21 BPs
- Create 5 client nodes
- Corrupt the source code located in ’chain_plugin.go’
  - Line 360-363
    - if(!chain.is_known_unexpired_transaction (id)){...}
  - Line 354-358
    - auto id = trx->id();
    - if(fc::time_point(trx->expiration()) >=block_time){...}
- Kill nodeos
- Reinitiate nodeos using corrupted source code
- Check to see if the block producer with the corrupted source code can participate in the network

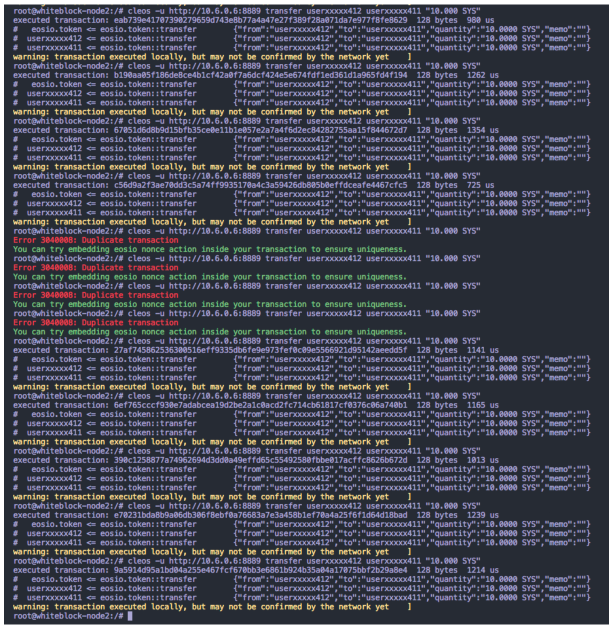
c) Results:

In this network, one of the block producers was configured with client software that blocks all of the transactions that are processed. Block producers process transactions following a preset schedule in a round robin fashion. Transactions were constantly sent in this network and whenever the block producer with corrupted software is activated to process transactions, errors were noted in this system.

Fig. 18 indicates that transactions are constantly being processed in this network as indicated by the node: <warning: transaction executed locally, but may not be confirmed by the network yet]>. Though it is important to note the error pertaining to duplicate transactions that emerge in the network. The duplicate transaction errors execute every time the block producer with faulty code processes transactions. After the duplicate transaction errors, the non faulty block producers continue to successfully process transactions. One important point to make here is that only the block producer that is running the modified source code is able to notice the duplicate transaction errors. Nothing in the system allows other block producers to audit the logs of the faulty block producer because the system relies on social trust that each block producer is correctly running the right software and there is no formal verification of that software.

d) Analysis:

This is a severe vulnerability in the system. If block producers do not need to reveal the code that they are running, the block producers can alter their systems to act in a nefarious manner. Block producers have the ability to maliciously



```
root@whiteblock-node2:/# cleos -u http://10.6.0.6:8889 transfer userxxxxx412 userxxxxx411 "10.000 SYS"
executed transaction: eab739e41707390279659d743e8b77a4a47e27f389f28a071da7e977f8fe8629  128 bytes  980 us
#   eosio.token <= eosio.token::transfer        {"from":"userxxxxx412","to":"userxxxxx411","quantity":"10.0000 SYS","memo":""}
#  userxxxxx412 <= eosio.token::transfer        {"from":"userxxxxx412","to":"userxxxxx411","quantity":"10.0000 SYS","memo":""}
#  userxxxxx411 <= eosio.token::transfer        {"from":"userxxxxx412","to":"userxxxxx411","quantity":"10.0000 SYS","memo":""}
warning: transaction executed locally, but may not be confirmed by the network yet    ]
root@whiteblock-node2:/# cleos -u http://10.6.0.6:8889 transfer userxxxxx412 userxxxxx411 "10.000 SYS"
executed transaction: b190aa05f186de8ce4b1cf42a0f7a6dcf424e5e674fdf1ed361d1a965fd4f194  128 bytes  1262 us
#   eosio.token <= eosio.token::transfer        {"from":"userxxxxx412","to":"userxxxxx411","quantity":"10.0000 SYS","memo":""}
#  userxxxxx412 <= eosio.token::transfer        {"from":"userxxxxx412","to":"userxxxxx411","quantity":"10.0000 SYS","memo":""}
#  userxxxxx411 <= eosio.token::transfer        {"from":"userxxxxx412","to":"userxxxxx411","quantity":"10.0000 SYS","memo":""}
warning: transaction executed locally, but may not be confirmed by the network yet    ]
root@whiteblock-node2:/# cleos -u http://10.6.0.6:8889 transfer userxxxxx412 userxxxxx411 "10.000 SYS"
executed transaction: 67051d6d8b9d15bfb35ce0e11b1e057e2a7a4f6d2ec84282755aa15f844672d7  128 bytes  1354 us
#   eosio.token <= eosio.token::transfer        {"from":"userxxxxx412","to":"userxxxxx411","quantity":"10.0000 SYS","memo":""}
#  userxxxxx412 <= eosio.token::transfer        {"from":"userxxxxx412","to":"userxxxxx411","quantity":"10.0000 SYS","memo":""}
#  userxxxxx411 <= eosio.token::transfer        {"from":"userxxxxx412","to":"userxxxxx411","quantity":"10.0000 SYS","memo":""}
warning: transaction executed locally, but may not be confirmed by the network yet    ]
root@whiteblock-node2:/# cleos -u http://10.6.0.6:8889 transfer userxxxxx412 userxxxxx411 "10.000 SYS"
executed transaction: c56d9a2f3ae70dd3c5a74ff9935170a4c3a59426db805b0effdceafe4467cfc5  128 bytes  725 us
#   eosio.token <= eosio.token::transfer        {"from":"userxxxxx412","to":"userxxxxx411","quantity":"10.0000 SYS","memo":""}
#  userxxxxx412 <= eosio.token::transfer        {"from":"userxxxxx412","to":"userxxxxx411","quantity":"10.0000 SYS","memo":""}
#  userxxxxx411 <= eosio.token::transfer        {"from":"userxxxxx412","to":"userxxxxx411","quantity":"10.0000 SYS","memo":""}
warning: transaction executed locally, but may not be confirmed by the network yet    ]
root@whiteblock-node2:/# cleos -u http://10.6.0.6:8889 transfer userxxxxx412 userxxxxx411 "10.000 SYS"
Error 3040008: Duplicate transaction
You can try embedding eosio nonce action inside your transaction to ensure uniqueness.
root@whiteblock-node2:/# cleos -u http://10.6.0.6:8889 transfer userxxxxx412 userxxxxx411 "10.000 SYS"
Error 3040008: Duplicate transaction
You can try embedding eosio nonce action inside your transaction to ensure uniqueness.
root@whiteblock-node2:/# cleos -u http://10.6.0.6:8889 transfer userxxxxx412 userxxxxx411 "10.000 SYS"
Error 3040008: Duplicate transaction
You can try embedding eosio nonce action inside your transaction to ensure uniqueness.
root@whiteblock-node2:/# cleos -u http://10.6.0.6:8889 transfer userxxxxx412 userxxxxx411 "10.000 SYS"
Error 3040008: Duplicate transaction
You can try embedding eosio nonce action inside your transaction to ensure uniqueness.
root@whiteblock-node2:/# cleos -u http://10.6.0.6:8889 transfer userxxxxx412 userxxxxx411 "10.000 SYS"
executed transaction: 27af745862536300516eff9335db6fe9e973fef0c09e5566921d95142aeedd5f  128 bytes  1141 us
#   eosio.token <= eosio.token::transfer        {"from":"userxxxxx412","to":"userxxxxx411","quantity":"10.0000 SYS","memo":""}
#  userxxxxx412 <= eosio.token::transfer        {"from":"userxxxxx412","to":"userxxxxx411","quantity":"10.0000 SYS","memo":""}
#  userxxxxx411 <= eosio.token::transfer        {"from":"userxxxxx412","to":"userxxxxx411","quantity":"10.0000 SYS","memo":""}
warning: transaction executed locally, but may not be confirmed by the network yet    ]
root@whiteblock-node2:/# cleos -u http://10.6.0.6:8889 transfer userxxxxx412 userxxxxx411 "10.000 SYS"
executed transaction: 6ef765cccf930e7adabcea19d2be2a1c0acd2fc714cb61817cf0376c06a740b1  128 bytes  1165 us
#   eosio.token <= eosio.token::transfer        {"from":"userxxxxx412","to":"userxxxxx411","quantity":"10.0000 SYS","memo":""}
#  userxxxxx412 <= eosio.token::transfer        {"from":"userxxxxx412","to":"userxxxxx411","quantity":"10.0000 SYS","memo":""}
#  userxxxxx411 <= eosio.token::transfer        {"from":"userxxxxx412","to":"userxxxxx411","quantity":"10.0000 SYS","memo":""}
warning: transaction executed locally, but may not be confirmed by the network yet    ]
root@whiteblock-node2:/# cleos -u http://10.6.0.6:8889 transfer userxxxxx412 userxxxxx411 "10.000 SYS"
executed transaction: 390c1258877a74962694d3dd0a49effd65c55492580fbbe017acffc8626b672d  128 bytes  1013 us
#   eosio.token <= eosio.token::transfer        {"from":"userxxxxx412","to":"userxxxxx411","quantity":"10.0000 SYS","memo":""}
#  userxxxxx412 <= eosio.token::transfer        {"from":"userxxxxx412","to":"userxxxxx411","quantity":"10.0000 SYS","memo":""}
#  userxxxxx411 <= eosio.token::transfer        {"from":"userxxxxx412","to":"userxxxxx411","quantity":"10.0000 SYS","memo":""}
warning: transaction executed locally, but may not be confirmed by the network yet    ]
root@whiteblock-node2:/# cleos -u http://10.6.0.6:8889 transfer userxxxxx412 userxxxxx411 "10.000 SYS"
executed transaction: e70231bda8b9a06db306f8ebf0a76683a7e3a458b1ef70a4a25f6f1d64d18bad  128 bytes  1239 us
#   eosio.token <= eosio.token::transfer        {"from":"userxxxxx412","to":"userxxxxx411","quantity":"10.0000 SYS","memo":""}
#  userxxxxx412 <= eosio.token::transfer        {"from":"userxxxxx412","to":"userxxxxx411","quantity":"10.0000 SYS","memo":""}
#  userxxxxx411 <= eosio.token::transfer        {"from":"userxxxxx412","to":"userxxxxx411","quantity":"10.0000 SYS","memo":""}
warning: transaction executed locally, but may not be confirmed by the network yet    ]
root@whiteblock-node2:/# cleos -u http://10.6.0.6:8889 transfer userxxxxx412 userxxxxx411 "10.000 SYS"
executed transaction: 9a5914d95a1bd04a255e467fcf670bb3e6861b924b35a04a17075bbf2b29a8e4  128 bytes  1214 us
#   eosio.token <= eosio.token::transfer        {"from":"userxxxxx412","to":"userxxxxx411","quantity":"10.0000 SYS","memo":""}
#  userxxxxx412 <= eosio.token::transfer        {"from":"userxxxxx412","to":"userxxxxx411","quantity":"10.0000 SYS","memo":""}
#  userxxxxx411 <= eosio.token::transfer        {"from":"userxxxxx412","to":"userxxxxx411","quantity":"10.0000 SYS","memo":""}
warning: transaction executed locally, but may not be confirmed by the network yet    ]
root@whiteblock-node2:/# 
```

Fig. 18. Stream of Constant Transactions to Block Producers

alter block transactions without the consent from the rest of the network, due to the black box nature of this system. This particular vulnerability further indicates how the overall network does not have a viable consensus algorithm as the underlying infrastructure of the network is not configured as a blockchain, rather a network of non transparent data centers.

4) Transactions as Proof of Stake Vulnerability:

a) Overview:

- partition the network
- send different transactions to build different multi-index tables
- reintroduce the nodes
- send transactions from client nodes
- check if transactions have been included in the block

b) Results:

Limitations were discovered regarding this test. As discussed previously, the multi-index table acts as a mempool designed to contain unprocessed transactios. However, there exists no means of viewing the current state of the mempool, short of having the process running as a slave to a debugger with heap inspection. Counter intuitively, within this software architecture, only the software itself is able to access the multi-index of the network, leaving open source developers oblivious to the information on unprocessed transactions.

c) Analysis:

This lack of accessibility is uncharacteristic of open source software. Normally in an open source ecosystem, it would be expected that developers would have access to these types of databases. The expectation would be that developers should be able to access the mutli-index table in order to verify that transactions are legitimate. Unfortunately, as can be seen from the source code, there is no formal verification protocol that occurs to validate transactions beyond the fact that that block producers are including them within transactions. The configuration of this system introduces further suspicion as mentioned in our performance testing on the fate of orphaned transactions that are not successfully processed by the network. In Ethereum, orphaned and stale transactions are apparent for the network to see in the available mempool, while economics are used to determine whether miners or validators choose to process those transactions.

In EOS, there is no true mechanism to determine the fate of unprocessed transactions. One theory as mentioned in our performance section is that transactions are selectively chosen for the purpose of maximizing transaction throughput in the system while bypassing difficult transactions that would cause network congestion. Picking transactions in this manner does not alleviate network congestion but rather ignores it. What this means is that unprocessed transactions are kept in the computer heap while only easy transactions are successfully processed. Unprocessed transactions are essentially left behind as there is no access to the multi-index table that houses these transactions. This evidence further calls into question, the design of EOS and how it functions as a software that does not successfully optimize the consensus around processing the transactions that are sent to the network.

IV. Economic Systems Analysis

Within blockchain-based networks, the implementation of effective game theory design is paramount to the success of the system. The method in which these systems go about doing so is generally reliant on suitable token economics, which attempt to incentivize human behavior using some degree of economic reward. The EOS system attempts to provide such a model of its own and the following section illustrates the economics of the EOS token.

A. *Context of the EOS Token*

Prior to discussing the details of the EOS token economics, it is important to provide some context concerning the history of the EOS token, including the technologies involved in the system, the process of the token sale and its distribution, and the context of various stakeholders in the ecosystem. EOS emerged as a potential decentralized application platform in 2017, when the market leading decentralized application platform, Ethereum, was encountering significant performance and scaling challenges, including high fees, increased blockchain bloat, and sub-optimal transactions times which had a drastic effect on user experience.

The EOS platform was proposed by Dan Larimer, who has a history of developing different blockchain projects, most notably Bitshares and Steem.it. Much of the technical architecture of the EOS platform was first used in these platforms and systems, so we can make comparisons when appropriate. The primary carry over from Bitshares and Steem.it is Larimers commitment to the delegated proof-of-stake byzantine fault tolerance (DPoS-BFT) consensus mechanism for driving agreement between nodes. While the performance and fault tolerance of this system is discussed in other sections, we discuss how the DPoS-BFT system affects the EOS network.

1) Initial Token Sale: EOS emerged in the media as a force through the scale of its token sale. During its token sale, EOS raised $4 billion over the course of one year-the largest token sale to date. The tokens were initially sold as ERC-20 tokens on the Ethereum network. As the EOS network approached launch, Block.one, the group in charge of writing and building the EOS software executed a token swap for ERC-20 tokens for native EOS tokens [35]. The token swap was a critical component of the EOS main-net launch, as token holders who did not swap their tokens by June 2, 2018 lost the opportunity to redeem their tokens. Many exchanges agreed to automatically swap tokens for holders, which helped the token swap process; ultimately, only 0.3% of the tokens were left unaccounted for [36].

2) Distribution of Tokens Across Accounts: Within the context of the token sale, it is important to understand who owns the EOS tokens. The distribution of EOS token holders gives insight into the level of economic centralization of the system. As will be discussed later, this economic centralization deeply impacts the behavior of the system because EOS use of DPoS. It is against the EOS constitution for a member to hold more than 10% of the token supply [37]. However, this rule is difficult to enforce as one member could control multiple accounts. As of June 3, 2018 (at the time of the EOS main-net launch), the top ten addresses in the EOS network controlled 496 735 539 out of the 1 billion tokens (i.e. 49.67%), while the top 100 addresses held 74.82% of all tokens [38], [39]. From these numbers, it appears that EOS tokens are concentrated among a small number of addresses. Even after accepting that some of these addresses represent exchange wallets, this has consequences for the governance of EOS block producers, as discussed in Section IV.E.

3) Token Inflation: While 1 billion tokens were initially sold during EOS initial coin offering, the token also has a 5% annual inflation rate. The reason that EOS has such inflation is because the inflation is meant to replace transaction fees for users on the network. 1% of the inflation is given to block producers as payment for securing the network. Out of this one percent, 0.25% of the reward is given to the block producers, while the other 0.75% is awarded to the standby block producers who received more than 100 EOS token votes. The other four percent is deposited into a smart contract called the Worker Proposal Fund [40]. Supposedly, the Worker Proposal Fund is governed by EOS token holders to further develop the EOS ecosystem, though there is currently no process to do so. There has been some discussion in the EOS community about eliminating the Worker Proposal Fund and instead burning these tokens (which would keep the EOS token inflation to 1% annually). These discussions have been in the larger context of increasing deflationary forces in the EOS network: namely, now including a one percent fee on RAM market transactions [41].

B. EOS Token Utility

In this section, we will try to highlight the various utilities of the EOS token. While the cryptocurrency market is currently inflated with a lot of overvalued projects, in an efficient market, the value of various tokens would be tied to the fundamental utility of the respective tokens in their systems. Thus, understanding the fundamental utility of the token in the EOS network is important to understanding the value of the token.

Beyond understanding the economic value of a token, understanding the EOS tokens utility is important in analyzing the cryptoeconomic system surrounding the token, including the interaction and relationships between different stakeholders. In most cryptocurrrency systems, the incentives surrounding the systems native asset need to be designed to encourage various properties: security, censorship resistance, and decentralization. These three properties are deeply tied to each other. By understanding the EOS tokens utility, we can better understand its incentive structure to better evaluate how the system will evolve with respect to these properties.

1) Voting For Block Producers: One of the primary purposes of the EOS token is voting for block producers in the DPoS consensus system. In the Delegated Proof-of-Stake consensus mechanism, participants in the voting process use their tokens to signal which block producers they want to construct blocks in the next epoch. In every voting round, the 21 block producers with the highest number of tokens backing them. There is then a round robin between block producers. Each block producer constructs 6 blocks with a block time of 0.5 seconds. If a block producer misses a block, then an empty block is simply committed to the chain [2]. Voters are continuously staking tokens for the block producers, changing their votes if they do not like the block producer's behavior. After every block producer constructs their set of blocks, the votes are reevaluated to generate the net set of block producers. In this scheme, block producers solicit votes from token holders in order to be elected.

There are a number of key nuances with the voting system in EOS [42]. First, accounts can vote for at most 30 block producers. Second, the amount of votes an account gets is the number of staked tokens the account has, so unstaked tokens cannot be used to vote. Third, when an account votes for multiple block producers, the block producers receive votes for the entirety of the accounts balance, i.e the votes are not split up among the block producers. For example, if an account has 10 EOS token and votes for 3 different block producers, then each of those three block producers receives 10 votes.

2) Delegation of Resources: The second critical utility that the EOS token provides is to delegate resources of the network. The EOS whitepaper identifies three distinct categories of resources in blockchain systems: bandwidth and log storage, computation and computational backlog, and state storage [2]. Bandwidth is a log of all the different Actions that have occurred and is measured by the rate of bytes being added to the ledger; from this log, the present state of the system can be reconstructed. The computational metric is measured as the amount of CPU time that can be billed (a maximum of 400 ms per second, with an average of 40 ms per second on the EOS network [43]. The CPU time is delegated to accounts proportional to the amount of tokens the accounts have staked. The final resource is state storage, referred to as RAM in the EOS whitepaper, which represents the information that is required for various application logic [2]. To distribute these resources, the EOS protocol utilizes the EOS token. According to the whitepaper, if an account holds 1% of the total EOS tokens in circulation, then the account is entitled to 1% of the networks bandwidth, computation, and state storage (RAM).

One important characteristic of resources in the EOS network is how bandwidth and computation can be rented out by token holders. This means if a token holder is not using the entirety of their allocated bandwidth and computation, then they can delegate or rent the extra resources to other users on the network. However, unlike bandwidth and computation, RAM cannot be delegated. Instead, RAM can be purchased from an EOS system contract according to previously defined market rates [44]. The effects of these dynamics on the economics of EOS will be discussed further in Section IV.E.

3) Fees in the RAM Market: The final key use of the EOS token is that the EOS token is used to pay the 1% fee in the RAM market [41]. The fee is burned in the RAM market system contract. The intent of the fee is to offset the inflation in the system and disincentivize speculators in the RAM market.

C. Understanding EOS Supply And Demand

As with any economic system, it is important to understand the two most fundamental price drivers: supply and demand.

1) EOS Supply: A total of 1 billion tokens were released during the token sale. Most of this total was redeemed on the main-net, not all of the were, which slightly decreased the supply [36]. Second, there is the 5% inflation rate of the EOS token. One percent of this inflation is given to block producers (with .25% going to the elected block producers and .75% being divided by standby block producers who received votes) [40]. Third, the liquid supply of the EOS token depends on how many of the tokens are currently being staked for resources on the network. If an application developer is using these tokens to reserve resources, these tokens are essentially removed from the supply since the application developer is not going to sell these tokens. Fourth, EOS tokens are burned as a fee when RAM is bought from the EOS system contract that controls the RAM market [41]. Lastly, a fifth deflationary force on the EOS network is for the namespace auctions, in which holders of their tokens can purchase various domain names in the EOS network [40]. These tokens are burned and removed from supply.

The supply of the EOS token could also change should the change be approved through the EOS governance process. For this to occur 15/21 block producers would have to vote in favor of the change for 30 days before the change would be implemented [2].

2) EOS Demand: The demand for the EOS token is essentially driven by the various utilities discussed in Section IV.B and the incentives discussed later in Section IV.D. Thus, demand should be based off 1) who wants to vote for block producers (and how much are they willing to pay this right) and 2) who wants to access resources on the network (and how much of the resources they want). Beyond these two factors, the demand for the EOS token is also impacted by the overarching dynamics of the cryptocurrency market, in which speculators often rely on volatile boom-bust cycles for returns.

D. Incentive Structure of the EOS Token

This section will attempt to illustrate the different incentive structures of different stakeholders in the EOS network, specifically with respect to the EOS token. We identify 4 distinct stakeholders in the EOS system: users, application developers, block producers, and standby block producers. Each of these different stakeholders interacts with the EOS token in a different way.

1) Users: One of the primary goals of the EOS protocol is to eliminate the need for users to pay fees in order to use decentralized applications. This has been one of the larger points of friction involved in the user experience of Ethereum-based decentralized applications, which implement a gas model that holds users responsible for paying for the computational overhead required to execute transactions and other actions on the Ethereum blockchain . This gas fee also helps prevent spam attacks on the network and creates a hard limit on the complexity of smart contracts which prevents malicious code from stalling the network (infinite loops, brute force attacks, etc). EOS attempts to address this pain point by eliminating the need for transaction fees on behalf of the user. To some extent, however, this further reduces the incentive for users to hold tokens.

The primary incentive for users to hold EOS tokens is the ability to delegate authority to the block producers they choose to act on their behalf in the process of validating transactions and blocks, however, this can be undermined by the block producers themselves since there is no deterministic mechanism to prevent their collusion or ability to vote for themselves.

2) Application Developers: In the EOS network, the developers are responsible for maintaining the costs associated with running their applications and in doing so, the EOS system is able to provide a user experience free of service-related fees [2]. This mechanism implies a strong incentive for application developers to hold and stake EOS tokens in exchange for access to necessary resources which ensure low-friction interactions with their applications. Alternatively, application developers can require users to stake their own tokens in order to interact with their applications. This is similar to Ethereum's gas model which places the responsibility of compensating for the cost of interacting with the application on the users.

This paradigm, however, can imply a dangerous Web 2.0 model: users are therefore beholden to application developers in order access their data or state associated with the application, as it is under the control of the application developer who can choose which model to implement, which also becomes a form of application level censorship that the EOS economic model permits and even encourages.

3) Block Producers: Because block producers are voted into position using EOS tokens, there is a strong incentive for block producers to hold EOS tokens so that they can vote for themselves. There is also a strong incentive for block producers to solicit the votes of EOS token holders to further ensure their position as a block producer. This voting process introduces a political dynamic that is discussed further in Section IV.E.

4) Standby Block Producers: Because only 21 block producers are selected, there remains a large set of people who want to be block producers but are stuck on the outside looking in. These are groups who either dont hold enough EOS tokens to vote themselves into being block producers or have failed to solicit enough votes. These standby block producers are critical to the EOS DPoS model for a number of reasons.

Standby block producers create competition for existing block producers. If a block producer behaves badly or does not perform their duties, standby block producers are there to take their place. Second, standby block producers act as a fault tolerance mechanism; if a block producer does not produce blocks for a 24-hour period, a standby block producer should replace them. Standby block producers are incentivized to continue lobbying votes because a portion of the inflation reward generated every block is divided between them.

E. Voting and Governance of the EOS Block Producers

As explained in Section IV.B.1, one of the fundamental utilities of the EOS token is voting for block producers. This functionality emerges because of EOS use of Delegated Proof-of-Stake, in which token holders vote for who they want to create blocks. However, higher levels of governance, including changes to the protocol, are mediated through block producers [2]. Thus, the voting process for block producers is also the mechanism through which the EOS network is governed with respect to censorship and upgrades.

Token holders continuously vote for block producers. Every epoch (126 blocks), the top 21 block producers are selected for the next epoch. By winning the election, block producers are gifted immense power in the system: 1) they can vote to freeze accounts, 2) they can change malicious account code, and 3) they vote to make changes and upgrades to the protocol [2]. Because the voting process is directly tied to the economics of the system, understanding the dynamics of the EOS token value, supply, and incentives gives insight into how the network is governed.

It is clear that the EOS network is censorable, as the whitepaper explicitly states that block producers can agree to freeze accounts change malicious code [2]. We have already seen accounts be frozen, showcasing how the network has a governance structure that enables censorship [45] This is because the EOS protocol does not have algorithmic, codified defenses to malicious actors. Instead, the EOS protocol relies on the social and benevolent governance of selected block producers for security. By extension, the security of the protocol is based on the robustness of the EOS voting process.

Understanding EOS governance is closely connected to token distribution. In the EOS network, consensus is reached (on blocks, actions, and upgrades), when 15 out of the 21 block producers agree [2]. This provides the foundation for any analysis of EOS governance. Because EOS tokens are highly concentrated amounts in few accounts, a small number of accounts control the voting process for block producers. In fact, there are certain accounts that can guarantee their position as a block producer based off their EOS token holdings.

An even greater concern is large token holders working together solidify their positions, making it even more difficult for malicious block producers to be voted out. If various token holders communicate with each other, then they coordinate various actions on the network, including the censorship of accounts and upgrades to the protocol that reinforce their own authority. However, beyond just the theoretical collusion of block producers, there is evidence that block producers have been trading votes with each other to do just this: maintain their role as block producers. In the alleged reports, a major block producer and token holder, Huobi, has offered to vote for other block producers in return for some share of the block producers profits [46]. Notably, such behavior is against the EOS constitution [37]. However, we have not seen the accounts involved in such cartelization be frozen, bringing up questions about the fairness of the EOS governance process.

The governance system of EOS is fundamentally dependent on block producers being held accountable. If a block producer is not providing adequate resources or honoring the protocol, then token holders must be able to vote them out. If a block producer is unfairly censoring transactions or smart contracts, then the token holders must be able to vote them out. If block producer is supporting an upgrade or change to the network that token holders do not support, then token holders must be able to vote them out. The core assumption in the EOS security model is that token holders have the ability to vote bad actors out of their position of power. Block producers can work together to ensure their authority in the network, which fundamentally contradicts the requirements for a secure, decentralized, and censorship resistant smart contract platform.

F. The Resource Market on the EOS Network

1) Overview of the Resource Market: One of the important economic features of the EOS system is the market for resources. The EOS uses a different model for resource allocation than Ethereum. In the EOS model, the three resources (bandwidth, computation, and RAM) are allocated based on how much EOS token a particular account has. The block producers are in charge of allocating resources proportional to the amount of token claiming a particular resource. As discussed earlier, token holders can delegate or rent out the extra bandwidth or computation they are entitled to; however, this is not the case for RAM [2].

In the EOS resource markets, resources are rented. The goal is that longer term renting contracts reduce volatility in prices, decrease short term denial of service attacks due to high fees, and are easier for economic accounting [43]. However, this model creates its own challenges. First, the liquidity of the EOS token decreases when tokens are locked up in rent contracts. With lower liquidity, price discovery for the EOS token (and ultimately the underlying assets of bandwidth and computation) becomes more difficult. Second, the rent model opens up the resource markets to rent seeking by large token holders. Given the concentration of EOS tokens among a small number of accounts, rent seeking is an even larger concern: because a small number of accounts control so many EOS tokens, market prices of the resources can be controlled by a small number of actors in the system.

The problem of speculation and market manipulation has been especially pronounced in the RAM market. We have already seen block producers drive the price of RAM up through hoarding for their own profit [47]. By hoarding RAM, token holders can essentially drive the price up to extract money from those trying to access this resource. This speculation around the RAM market has made the price of RAM unaffordable, making it difficult for decentralized applications to operate [48]. To solve this problem, the EOS team implemented a Bancor style pricing algorithm. The Bancor Relay Tokens is a type of smart contract that ensures that the exchange rate of two assets never reaches zero; when the supply of one asset is decreased, the exchange rate with respect to that asset increases [49].

Block.one and the EOS development team have implemented this algorithm for the RAM market. The system contract acts as a manager of the RAM resource to theoretically aid in price discovery. This system contract enables token holders to buy and sell RAM back at previously specified market rates [44]. In order to limit speculation, every transaction with the system contract burns a 1% EOS token fee [41]. However, it is unclear if this new algorithmic pricing system effectively curbs speculation as the dynamics around speculation still exist. If the exchange rate of the Bancor Relay is low, then token holders will hold on to the RAM they have already purchased until the price of RAM goes up. This behavior is again more pronounced when a few token holders can buy up a lot of RAM when the price is low. While the 1% fee theoretically limits speculation, this actually shifts the balance of power in the market even more towards block producers, as they are the ones who can afford to pay the 1% fee because they are earning EOS tokens as a reward.

2) Understanding the Effect of the RAM Bancor Algorithm: Inspection of the EOS RAM generation contract yielded insight into how RAM is created. From looking at the source code, the supply of RAM is adjusted in terms of availability based on the the price of where RAM is traded according to the Bancor relay connector algorithm [50]. However, the actual supply of RAM is more based on the computational resources of the hardware used to run the network, rather than what the algorithm says. There is an attempt to algorithmically set the price of RAM based on a predefined formula, but there are situations in which the speculation on the price of computation in the network can cause the promised supply of the computational resources to exceed the actual limits of the hardware because the economics of algorithm are decoupled from the computational capabilities of the hardware.

The capacity of the EOS network is defined by the computational capacity of the EOS block producer hardware. According to the protocol, block producers publish their total computational capacity to the network so that token holders can use this information to vote and to buy and stake tokens appropriately. However, this introduces two main challenges: 1) it is difficult to verify that a block producer actually has the computational resources they claim to have (especially when the network is not under stress) and 2) it is difficult to verify that a block producer is allocating the computational resources fairly and according to the protocol. All of this asymmetry means that price discovery for these resources is difficult, if not impossible. Considering the blackbox nature of how the EOS block producer hardware is managed and implemented, it is nearly impossible to determine a fair market value for the RAM produced, which similarly affects the price discovery of EOS tokens.

According to the protocol, block producers should divide computational resources based on staked tokens. However, it is difficult to confirm that they are doing this because there is no way to know if what code the block producers are running. This contrasts starkly with Ethereum. The Ethereum protocol defines a strong set of rules that nodes must adhere to; because these rules are so strong, if a node does not adhere to the rules, they cannot participate in the network. This is what enables different nodes to run different code. However, the EOS whitepaper does not have the same detailed specifications. Similarly, the strict rules of the Ethereum protocol are cryptographically verifiable through Merkle proofs. The problem with the EOS protocol is that much of the behavior of block producers cannot be verified computationally or algorithmically. This highlights two major concerns with the EOS protocol. One, as discussed earlier, the EOS protocol relies on social governance to deal with bad actors. Second, this adds another layer of inefficiency (particularly with respect to information access) on top of the various market inefficiencies the cryptocurrency market already deals with.

3) Game Theory and Cryptoeconomic Mechanism Design: As described in the Incentive Structure of the EOS Token section, the primary incentive for owning EOS tokens revolves around the accumulation of value from token price appreciation as well as speculation of the value of computational resources that are generated from staking EOS tokens. Additionally there is an incentive for block producers to accumulate tokens for the purposes of maintaining their position as block producers through the voting process. These mechanisms are designed to encourage token holders to purchase tokens, though there are no internal feedback mechanisms designed for speculators of the RAM market to act fairly and ensure the health of the network itself.

Conversely in Ethereum, transaction fees are designed to incentivize miners to prioritize certain transactions over others. Higher gas prices result in miners making more money, though at the same time, the fees prevent spam transactions as attackers cannot spam the network without expending large monetary resources [51]. The system was designed to incentivize miners to both accumulate fees while also coordinating that same action toward securing the network for validation of transactions. In this ecosystem, there exists a positive feedback loop in which there is a coupling between a validators desire to secure the network as well as to generate fees [52]. As EOS lacks a mechanism designed to couple the desire to help the network while concurrently accumulating value toward self interest, it can be concluded that EOS does not secure the network using game theory incentive structures.

V. CONCLUSIONS

A. Architecture

Based on a thorough assessment of the entire system, EOS was originally architected as a blockchain, though the resulting platform failed to achieve the necessary composition of a blockchain system. The platform was instead, built as, a distributed database system. This becomes apparent from the substantial reliance on architecture where cryptographic validation is not necessarily instantiated, though rather implemented in proximity.

B. Performance

As observed in the section on performance, the transaction throughput in the system does not exceed 250 TPS even in optimal settings with 0 ms of latency and 0% packet loss. During tests with real world conditions of 50 ms of round trip latency and 0.01% packet loss, performance dropped below 50 TPS putting the system in close proximity to the performance that exists in Ethereum. Consensus in EOS does not exhibit Byzantine Fault Tolerance. The use of 21 block producers in the system is arbitrary as the producers are processing transactions in a round robin manner, with no algorithmic logic when block producers behave nefariously. Various forms of consensus failure were shown in the network testing. Consensus is entirely dependent upon the voting mechanism which is based on social consensus, not algorithmic BFT consensus.

C. Economics

An analysis of the EOS token system reveals that the network acts as a marketplace for available computation. The network provides computational services that are essentially credits for computation generated by the 21 block producers. Because the configuration and processing capabilities of the 21 block producers are shielded from the public, there is no way to discern the amount of computation available in the network that is being promised by the tokens. This black box nature of the block producers creates a system where unverified computational value is being introduced into the system, creating the potential for instability.

D. Summary

From this comprehensive analysis of the EOS system, it has become apparent that in order for EOS to be able to successfully act as a foundational base layer protocol, it needs to re-architect a significant portion of its infrastructure. EOS can potentially act as a side chain appended to other more foundationally secure networks, though the system would need to be rebuilt in order to address the problems detailed in this report.

ACKNOWLEDGMENT

A number of different people from various organization assisted in this research. We would like to thank Daniel Choi and Trey Zhong from the Whiteblock team; Shahan Khatchadourian, Ben Edgington, and Nicolos Lioshon from the PegaSys team at Consensys; Wayne Chang, Grec Rocco, and Steve Gattuso from the Alpine team at ConsenSys; and Antione Toulme from the ConsenSys Core Engineering team.

We would also like to thank Andreas Freund, John Wolpert, Ray Valdes, and Joe Lubin from ConsenSys for additional contributions.

This research was also supported and reviewed by members of the Duke Blockchain Lab at Duke University.

Lastly, we'd like to thank the following groups for their committed support to this open ecosystem research effort: Cosmos, the Ethereum Community Fund, IMToken, Amis, Status, Maker Dao, Transference Fund, 1KX, CoinFund,

the Enterprise Ethereum Alliance, Loom, Grid+, PlaTON, MixLabs, Web3 Foundation, Ledger Capital, Microsoft, Google, ConsenSys, and Bo Shen (Founding partner of Fenbushi Capital and Co-founder of Bitshares).

References

[1] M. Bolton. ”ICO New: Investments Soar Past the $12 Billion Mark as 2018 Becomes the Year of ICOs”. Internet: https://theindependentrepublic.com/2018/08/10/ico-news-investments-soar-past-the-12-billion-mark-as-2018-becomes-the-year-of-icos/, Jul. 2018 [Oct. 26, 2018].

[2] ”EOS.IO Technical Whitepaper v2”. Internet: https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md, Apr. 28, 2018 [Oct 27, 2018].

[3] Whiteblock, Inc. Internet: https://www.whiteblock.io/

[4] EOS Developer Documentation. ”Home”. Internet: https://developers.eos.io/eosio-home/docs, [Oct. 30, 2018].

[5] EOS Developer Documentation. ”Nodeos”. Internet: https://developers.eos.io/eosio-nodeos/docs/, [Oct. 30, 2018].

[6] EOS Developer Documentation. ”Cleos”. Internet: https://developers.eos.io/eosio-nodeos/docs/cleos-overview, [Oct. 30, 2018].

[7] EOS Developer Documentation. ”Keos”. Internet: https://developers.eos.io/keosd/docs/, [Oct. 30, 2018].

[8] EOS Developer Documentation. ”Installing the Contract Development Toolkit”. Internet: https://developers.eos.io/eosio-home/docs/installing-the-contract-development-toolkit, [Oct. 30, 2018].

[9] EOS Developer Documentation. ”Accounts”. Internet: https://developers.eos.io/eosio-home/docs/accounts-1, [Oct. 30, 2018].

[10] EOS Developer Documentation. ”Learn About Wallets, Keys, and Accounts With Cleos”. Internet: https://developers.eos.io/eosio-nodeos/docs/learn-about-wallets-keys-and-accounts-with-cleos, [Oct. 30, 2018].

[11] Genereos. ”Name Bidding and Premium Names on EOS”. Internet: https://steemit.com/eos/@genereos/name-bidding-and-premium-names-on-eos, May, 2018 [Oct. 30, 2018].

[12] EOS Developer Documentation. ”Accounts and Permissions”. Internet: https://developers.eos.io/eosio-nodeos/docs/accounts-and-permissions, [Oct. 30, 2018].

[13] B. Xu. ”Blockchain vs. Distributed Ledger Technologies Part 2: Governing Dynamics”. Internet: https://media.consensys.net/blockchains-vs-distributed-ledger-technologies-part-2-governing-dynamics-a697848d5b82, May 23, 2018 [Oct. 30, 2018].

[14] Bluabaleno. ”EOS Whitepaper walk-through: Accounts and Action and Handlers”. Internet: https://steemit.com/eos/@bluabaleno/eos-whitepaper-walk-through-accounts-and-action-and-handlers Apr. 2018 [Oct. 30, 2018].

[15] Stack Overflow. ”How Does WASM Get Interpreted by the EOS Virtual Machine”. Internet: https://eosio.stackexchange.com/questions/167/how-does-wasm-get-interpreted-by-the-eos-virtual-machine, May 10, 2018 [Oct. 30, 2018].

[16] Web Assembly Github. Internet: https://github.com/WebAssembly/design/blob/master/Modules.md, [Oct. 30, 2018].

[17] EOS Developer Documentation. ”Communication Model”. Internet: https://developers.eos.io/eosio-cpp/docs/communication-model, [Oct. 30, 2018].

[18] m-i-k-e. ”The EOS ERC-20 Contract Explained”. Internet: https://steemit.com/eos/@m-i-k-e/the-eos-erc-20-contract-explained, 2017 [Oct. 30, 2018].

[19] EOS Developer Documentation. ”Action”. Internet: https://developers.eos.io/eosio-cpp/reference#action-1, [Oct. 30, 2018].

[20] EOS Developer Documentation. ”Transaction”. Internet: https://developers.eos.io/eosio-cpp/reference#transaction, [Oct. 30, 2018].

[21] EOS Developer Documentation. ”Sending an Inline Transaction to External Contract”. Internet: https://developers.eos.io/eosio-home/docs/sending-an-inline-transaction-to-external-contract, [Oct. 30, 2018].

[22] Blockgeeks. ”What is Ethereum Gas: Step-By-Step Guide”. Internet: https://blockgeeks.com/guides/ethereum-gas-step-by-step-guide/, [Oct. 30, 2018].

[23] B. Xu. ”Blockchain vs. Distributed Ledger Technologies”. Internet: https://media.consensys.net/blockchain-vs-distributed-ledger-technologies-1e0289a87b16, Apr. 5, 2018 [Oct. 30, 2018].

[24] Infinite X Labs. ”The Ultimate End to End EOS Dapp Development Tutorial Part 1”. Internet: https://medium.com/infinitexlabs/the-ultimate-end-to-end-eos-dapp-development-tutorial-part-1-2f99c512086c, Apr. 30, 2018 [Oct. 30, 2018].

[25] G. Wood. ”Ethereum Yellow Paper”. Internet: https://github.com/ethereum/yellowpaper, [Oct. 30, 2018].

[26] EOS Developer Documentation. ”Data Persistence”. Internet: https://developers.eos.io/eosio-home/docs/data-persistence, [Oct. 30, 2018].

[27] IBM Knowledge Center. ”Exclusive Locks and Shared Locks”. Internet: https://www.ibm.com/support/knowledgecenter/en/SSGMCP_5.5.0/applications/designing/dfhp39o.html, [Oct. 30, 2018].

[28] Boost Multi Index Library. Internet: https://www.boost.org/doc/libs/1_62_0/libs/multi_index/doc/index.html, [Oct. 30, 2018].

[29] EOS Developer Documentation. ”DB API”. Internet: https://developers.eos.io/eosio-cpp/docs/db-api, [Oct. 30, 2018].

[30] EOS Developer Documentation. ”Boot Sequence”. Internet: https://developers.eos.io/eosio-nodeos/docs/boot-sequence, [Oct. 30, 2018].

[31] EOS Github. Internet: https://github.com/EOSIO/eos, [Oct. 30, 2018].

[32] EOS Developer Documentation. Internet: https://developers.eos.io/, [Oct. 30, 2018].

[33] EOS New York. Internet: https://www.eosnewyork.io/tech, [Oct. 30, 2018].

[34] H. Finney. ”RPOW- Reusable Proofs of Work”. Internet: https://cryptome.org/rpow.htm, Aug. 16, 2004, [Oct. 30, 2018].

[35] G. Jenkinson. ”Moment of Truth for EOS: What’s Next for $4 bln EOSIO Following Launch of v1.0”. Internet: https://cointelegraph.com/news/moment-of-truth-for-eos-whats-next-for-4-bln-eosio-following-launch-of-v10, Jun. 5, 2018 [Oct. 26, 2018].

[36] B. Smith. ”EOS token swap: What to do if you missed registration”. Internet: https://www.coininsider.com/what-to-do-if-you-missed-eos-token-registration/, Jun. 5, 2018 [Oct. 26, 2018].

[37] ”EOS Constitution”. Internet: https://github.com/EOSIO/eos/blob/5068823fbc8a8f7d29733309c0496438c339f7dc/constitution.md, Jun. 1, 2018 [Oct. 26, 2018].

[38] ”Just 10 Addresses Hold Nearly 50 EOS Tokens”. Internet: https://www.trustnodes.com/2018/06/03/just-10-addresses-hold-nearly-50-eos-tokens, Jun. 3, 2018 [Oct. 26, 2018].

[39] D. Sui, J. Pfeffer, J. Gillis, E. Muzzy. ”A Retrospective of the EOS Token Sale”. Internet: https://media.consensys.net/a-retrospective-of-the-eos-token-sale-172d3437932b, Oct. 25, 2018 [Oct. 26, 2018].

[40] EOS Canada. ”Inflation on EOS: What do I need to know?”. Internet: https://steemit.com/eos/@eos-canada/inflation-on-eos-what-do-i-need-to-know, May 2018 [Oct. 26, 2018].

[41] D. Larimer. ”Introducing EOSIO Dawn 4.0”. Internet: https://medium.com/eosio/introducing-eosio-dawn-4-0-f738c552879, May 4, 2018 [Oct. 26, 2018].

[42] J. Kauffman. ”Everything You Need To Know About Voting On EOS”. Internet: https://www.eoscanada.com/en/voting-faqs, Jun. 12, 2018 [Oct. 26, 2018].

[43] D. Larimer. ”Proposal for EOS Resource Renting and Rent Distribution”. Internet: https://medium.com/@bytemaster/proposal-for-eos-resource-renting-rent-distribution-9afe8fb3883a, Aug. 2, 2018 [Oct. 26, 2018].

[44] D. Larimer. ”EOSIO RAM Market and Bancor Algorithm”. Internet: https://medium.com/@bytemaster/eosio-ram-market-bancor-algorithm-b8e8d4e20c73, Jul. 4, 2018 [Oct. 26, 2018].

[45] D. Floyd. ”EOS’ Blockchain Arbitrator Orders Freeze of 27 Accounts”. Internet: https://www.coindesk.com/eos-blockchain-arbitrator-orders-freeze-of-27-accounts/, Jun. 22, 2018 [Oct. 26, 2018].

[46] ”Rampant Collusion in EOS Exposed by Huobi Leak”. Internet: https://www.trustnodes.com/2018/09/29/rampant-collusion-in-eos-exposed-by-huobi-leak, Sept. 29, 2018 [Oct. 26, 2018].

[47] A. Ancheta. ”EOS Block Producer Traded RAM To $600k Profit - And Did Nothing Wrong”. Internet: https://cryptobriefing.com/eos-block-producer-ram-profit/, Jul. 9, 2018 [Oct. 26, 2018].

[48] A. Vaskevicius. ”EOS Stumbles Once More as Speculation Causes RAM to Spike Beyond Affordability”. Internet: https://toshitimes.com/eos-stumbles-once-more-as-speculation-causes-ram-prices-to-spike-beyond-affordability/, Jun. 2018, [Oct. 26, 2018].

[49] ”How do Relay Tokens work?”. Internet: https://support.bancor.network/hc/en-us/articles/360000471472-How-do-Relay-Tokens-work- [Oct. 26, 2018]

[50] Internet: https://github.com/EOSIO/eos/tree/master/contracts/bancor, May 22, 2018 [Oct. 26, 2018].

[51] N. Tomaino. ”Cryptoeconomics 101”. Internet: https://thecontrol.co/cryptoeconomics-101-e5c883e9a8ff, Jun. 4, 2018 [Oct. 26, 2018].

[52] Walking Tree Technologies. ”Understanding Gas in Ethereum”. Internet: https://medium.com/coinmonks/understanding-gas-in-ethereum-53ad816f79ae, Jul. 16, 2018 [Oct. 26, 2018].