Blockchain Consensus: An analysis of Proof-of-Work and its applications.

Amitai Porat[1], **Avneesh Pratap**[2], **Parth Shah**[3], **and Vinit Adkar**[4]

[1]aporat@stanford.edu
[2]avneeshp@stanford.edu
[3]parth95@stanford.edu
[4]vadkar@stanford.edu

ABSTRACT

Blockchain Technology, having been around since 2008, has recently taken the world by storm. Industries are beginning to implement blockchain solutions for real world services. In our project, we build a Proof of Work based Blockchain consensus protocol and evauluate how major applications can run on the underlying platform. We also explore how varying network conditions vary the outcome of consensus among nodes. Furthermore, to demonstrate some of its capabilities we created our own application built on the Ethereum blockchain platform. While Bitcoin is by and far the first major cryptocurrency, it is limited in the capabilities of its blockchain as a peer-to-peer currency exchange. Therefore, Ethereum blockchain was the right choice for application development since it caters itself specifically to building decentralized applications that seek rapid deployment and security.

1 Introduction

Blockchain technology challenges traditional shared architectures which require forms of centralized governance to assure the integrity of internet applications. It is the first truly democratized, universally accessible, shared and secure asset control architecture. The first blockchain technology was founded shortly after the US financial collapse in 2008, the idea was a decentralized peer-to-peer currency transfer network that people can rely on when the traditional financial system fails. As a result, blockchain largely took off and made its way into large public interest.

We chose to investigate the power of blockchain consensus algorithms, primarily Proof of Work. We wanted to see how this algorithm performs under a variety of network conditions as well as highlight its inner workings. Additionally we developed an application for a potential real-life use case of blockchain technology. We built our application on the Ethereum blockchain for a number of reasons. First, Ethereum for the time being, also relies on Proof of Work consensus and we wanted to build on top of a blockchain that closely replicates the consensus algorithms we examined. Second, even though it has much room left to grow, the Ethereum platform has a strong developer community which was able to provide sufficient support to bootstrap an application in a short amount of time.

Section 2 of this paper provides a high level overview of the proof of work consensus algorithm followed by a detailed description of our implementation. Additionally, we discuss the system behavior under varying conditions which we were able to control. We then discuss the performance results that we analyzed in 2.3. Beyond the consensus algorithm implementation we built a simple graphic user interface that allows for better understanding of network state in the simulation, which we briefly describe in 2.4. In Section 3, we discuss smart contracts, which are an essential construct on Ethereum. The smart contract overview provides necessary the background for our application, Lockbox, which is a secure storage service for personal information on the Ethereum blockchain.

2 Blockchain Consensus Protocol

In an open, trustless and distributed setting adversaries can act maliciously in many ways. For example, an attacker can make multiple pseudonyms and do a sybil attack[1], and hence making the typical quorum consensus techniques popular in distributed systems unapplicable. Bitcoin[2] uses proof-of-work protocol to reach consensus, which requires a node to try and solve a hard computational problem in order to validate a batch of transactions and add them as a new block to the blockchain. Another proposition is to use proof-of-stake[3] where user votes are weighted by their 'stake' and users might be penalized for faulty behavior. The following article[4] provides an overview of some more consensus protocols we do not discuss. We chose to examine proof-of-work due to it currently being the most widely adopted.

To understand the consensus protocol in more depth we decided to implement a proof-of-work consensus from scratch using Python. In a proof-of-work protocol, the complexity of the computational problem determines factors like the number of forks in the chain and the average time to mine a block. We present an analysis of how these parameters vary with the computational complexity. To validate our consensus protocol we implemented a simple transaction ledger which maintains a history of transactions. We implemented both unspent transaction output (UTXO) in Bitcoin, and Account based transactions as used in Ethereum.

Additionally, we provide a visualization tool which provides real time status of the blockchain as well as a per-node view of the distributed ledger.

2.1 Background

Bitcoin[2] uses proof-of-work to ensure that all participating nodes agree on the same branch of the blockchain. The main idea behind the protocol is to have nodes solve a computationally expensive problem before they can suggest a new block. The node which first solves the problem, mines the new block and broadcasts the message to the other nodes in the network who can easily verify its correctness based on the values in the block. This is because the hash values of new blocks in the blockchain are dependant on all previous hash values in the blockchain. If an attacker tries to modify a single block in the history, that would change the blockhash and make the whole subsequent blockchain invalid. In the rare case where multiple users verify a block simulatneously, the network temporarily forks until future blocks are mined at which point all nodes will converge on the current longest list of blocks.

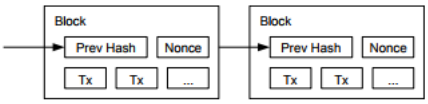

Figure 1. A part of Blockchain

2.2 System Overview

Our implementation allows to build networks consisting of any arbitrary number of nodes. Every node starts with a set of neighbors which it takes as an argument, but nodes can be added and removed from this list after startup. Each node maintains its own state of the blockchain, list of pending transactions, and a small set of neighbors. Additionally each node serves as a miner in the network. Miners continuously attempt to solve the proof-of-work in order to mine new blocks and be the first to extend their blockchain. When a miner successfully mines a new block, he shares the information with his neighbors. The information sharing is done asynchronously and the miner immediately starts trying to mine the next block. Since the information is shared asynchronously, different nodes can receive information in different order. To replicate the network latency in the actual network, we introduce artificial latency while sharing information in the network. The network latency between 2 nodes is proportional to the difference in their port numbers(we have considered nodes running as separate processes on the machine and ports as proxy for their geographic location). To share this information, each node exposes several HTTP endpoints where the nodes can submit new information like transactions or new blockchains. These requests are also served asynchronously so as to not hamper the mining process. On receiving a new blockchain, the nodes validate the blockchain and if the

length of the new blockchain is longer than their current version the nodes update their copy. This merge process can uncommit some transactions and the nodes merging the blockchain, therefore, make a note of any such transactions and reinsert them back in the list of pending transactions. Nodes also serve other HTTP endpoints to expose their state information. These are used to implement the visualization tools built over the network.

We tested the consensus implemented by the system above by using it to maintain a distributed ledger. Different cryptocurrencies use different types of transactions. In our implementation, we considered UTXOs approach used by Bitcoin[2] and Accounts as used in Ethereum[56]. Both approached provide different trade-offs. We mention some of them in section 2.3.

2.3 Analysis of the System

Theoretically there could be thousands of nodes around the world simultaneously solving the same proof-of-work problem. The purpose of making the problem so difficult to compute is to reduce the chances of nodes computing at the same time and thereby reduce the chance of forks. Forks affect the reliability of blockchain, and therefore Bitcoin[2] keeps the block generation time around 10 minutes to counter it. Since this is an important concern in the system, in our implementation we wanted to test how the system behaves when we change the complexity of the computational problem. To provide a common ground for these tests, we kept the network latency constant for each experiment and allowed the blockchain to grow consistently to the same length(100) before making our observations which are summarized in 1

Problem Complexity [1]	Average time to mine a block(in seconds)	Stale blocks mined	Average fork length
2.10^{-4}	0.03	302	21.47
2.10^{-5}	0.32	219	8.27
10^{-6}	2.62	42	2.61
2.10^{-7}	8.6	13	1

Table 1. Analysis for different levels of problem complexity

We see that as we increase the problem complexity the time to mine new block increases which is an expected result but we observe that the increase is not linear. While the problem complexity increases 20 times, the increase in avg time taken to mine a block is just over 8 folds. We reason that this is because of the higher number of collisions and forking that occurs at lower complexity. We therefore estimate the number of stale blocks(the blocks which were mined but were discarded due to conflicts) mined in the system and we see that in case of lower problem complexity this number is significantly higher. When the probability of making a correct guess is 2.10^{-5}, over 200 hundred blocks were discarded which is twice as much the final blockchain length. This means although the system is capable of increasing the length at a faster pace, most of the work is discarded and therefore the increase in time in not linear with problem complexity.

We also examine the average fork length in each case and fork length increases drastically as the problem becomes simpler. Having a higher fork length results in reduced consistency in the system making it less reliable. The above analysis provides an insight into the decision to keep the block mining time around 10 minutes in Bitcoin. The problem is made significantly harder so as to reduce the probability of multiple nodes solving the problem at the same time and also provide enough time for the new mined block to be distributed across the network.

As mentioned in 2.2, we tested our consensus protocol by using it to maintain a distributed ledger. We tested the correctness by giving the same list transactions to a centralized server and reconciling the values with the one in our distributed ledger. The different approach of UTXOs and Account lead to different transaction and state sizes. In our small experiment with 500 valid transactions generated over 50 accounts, the UTXO state tracked a list of around 113 unspent transactions, which is significant proportion of the input transactions, while Account approach just needed to maintain 50 accounts. The UTXO transactions were also larger in size. But at the same time UTXO approach provides flexibility in running 2 transactions over unrelated unspent transactions related to the same account simultaneously providing option for more parallel transactions.

2.4 Visualization Tool

To monitor our network we built a visualization tool over our system. It exposes 3 endpoints to query different pieces of information:

- Network: To see the network layout of the current system
- Blockchain Status: To see the current state of blockchain as maintained by different nodes. This tool is useful to see forks and merges in the chain.
- Distributed Ledger: To see the ledger values maintained by different nodes. The values are color coded to ease the viewing in case of system forks and when all the nodes are not maitaining the same consistent view.

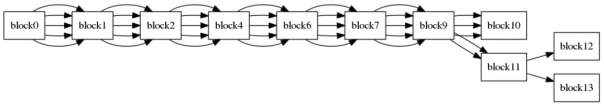

Figure 2. Tracking blockchain in the system

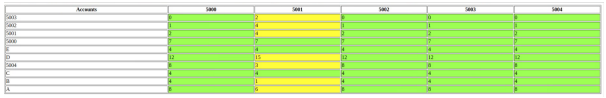

Figure 3. Constructing the Ledger

2.5 Future Work

Our current implementation is inspired from Bitcoin. However, Ethereum[5] uses a different computational problem, EthHash which rather than using the hashing abilities of the node focuses more on how fast the node can move data in its memory. To provide higher transaction rates and avoid forking in the network at the same time, Ethereum implements a simplified GHOST protocol introduced here[7]. It would be interesting to build a system inspired by design choices made in Ethereum and compare the performance with the current implementation. Proof-of-work consensus protocol is designed to be deployed in a trustless environment and it would be noteworthy to experiment with its performance by introducing faulty nodes into the environment and running transactions over an unreliable network.

3 Smart Contracts

A smart contract[6] is a computer program stored on the decentralized blockchain network that executes the terms defined inside of it. The contract only runs when it is invoked to do so by an external event or if some predefined condition is met. Smart contracts can be implemented using Solidity, Serpent, or LLL which are contract-oriented high level languages targeted for Ethereum Virtual Machine (EVM)[8]. EVM[8] is a Turing complete software that runs on the Ethereum network. It enables anyone to run any program written in any programming language given enough time and memory. For our implementation we chose Solidity to write our smart contracts. Solidity is statically typed, supports inheritance, libraries and complex user-defined types among other features. Figure 2 demonstrates how the solidity compiler (solc) generates byte level code for input smart contract which can be deployed using the EVM.

3.1 Smart Contract Applications

Smart contracts can: a) Manage agreements between parties, b) Provide utility to other contracts through inter contract communication c) Store information about an application, such as domain registration information or membership records. One of the simple use case of smart contract could be sending 5 ethers from account A to account B on a certain date. So in this case, user A would create a contract, and push the data to that contract so that it would know how much money to send and when. At the time of execution, the smart contract would verify that A has sufficient balance to execute before doing so.

Another more interesting application would be a crowdfunding contract. Wherein the contract would manage some desired price goal, and in return for their contributions people receive some sort of incentive, also in the contract. On Ethereum this frequently happens through an inital coin offering (ICO), where users submit ether to a contract and in return receive a new cryptocurrency.

To fully understand how smart contracts work on the blockchain and how they can be used to create decentralized applications we have implemented an application called Lockbox.

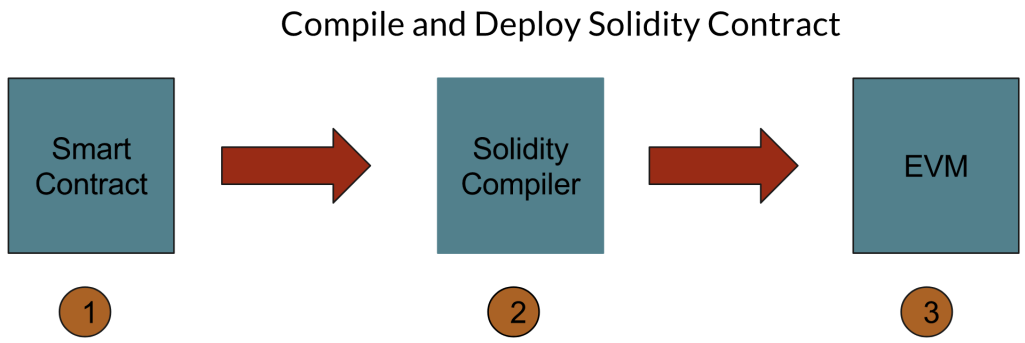

Figure 4. Steps to Deploy a smart contract on blockchain 1) create a smart contract in solidity, 2) compile and generate byte code and 3) deploy the contract on blockchain

4 Lockbox

Lockbox is a decentralized application for storing user information. Lockbox currently stores the following information through a smart contract: a) personal information such as SSN, license number and date of birth b) Login details for different services and c) credit card details for different accounts.

The lockbox application implementation has 2 abstractions: The backend is written in solidity, as a smart contract which lives on the network. The lockbox contract code has a collection of functions and state variables which reside at a specific address in the Ethereum blockchain once the contract is deployed on the blockchain. The frontend is implemented in Go, which provides an interface for the user and interacts with the necessary go bindings which are generated from smart contract creation.

In our decentralized lockbox application we provide a basic option to create a lockbox for users. We also provide two basic options to add and retrieve each kind of data/information from the blockchain network. Smart contract enable users of lockbox to request storage and retrieval of personal data from the blockchain network and validate storage proofs. Users can interact with the smart contract by sending transactions to the ledger that trigger function calls in the contract. For example: when a user creates a lockbox, it triggers a lockbox create function call in the contract and later this lockbox can be only accessed through user key. Smart contracts have a builtin ability to trace message senders, so when a user instantiates a lockbox, he is automatically set as the administrator from within the contract. Similarly, when an user adds or gets some information from the lockbox, it triggers lockbox add or get function calls in the smart contract for the transaction, each time verifying that the key from the request matches the key of the owner for the given information. This makes the storage of personal data secure as this information in the network can only be accessed by the owner of the lockbox through owner's user key. The security functionality is built in to the contract, something that Ethereum emphasizes so that certain applications can be deployed rapidly.

We ran our application on the Ethereum testnet to visualize how different operations in our lockbox application invoke transactions and how transactions get mined by some miner in the testnet. We also validated the security aspects of the application by experimenting the scenarios where user A attempts to add or retrieve data from user B lockbox but remains unsuccessful. In the future we would like to see if we can separate the lockbox contract from the registry of all lockboxes. This way we ensure that user contracts are completely independent. Overall we were very encouraged with the tools available for development on Ethereum.

5 Conclusion

Blockchain is a technology which has the potential to serve a vast array of internet applications in the future. Due to proof-of-work consensus, eventual consistency among all participating nodes is expected, and due to the difficulty of mining blocks, it is rare for the blockchain to fork based on two different mining results being broadcasted at the same time. Even in the case of a fork, there is a recovery mechanism, making proof of work safe overall.

The downside of this form of consensus is that it is very computationally expensive, and any network running the protocol is susceptible to manipulation if a participating miner has more than 51% of the computing power.

Our aim while implementing a proof-of-work based blockchain consensus was to gain a better understanding of the protocol. Through our work and experiments, we were able to achieve our set goal and also understand the reason behind key design aspect in Bitcoin. While building the visualization tool, our aim was to provide a simple interface for a user to keep track of the system and further supplement the understanding of the protocol. Demonstrating robustness in the presence of failure is critical to any viable system, and we hope that in the future we can analyze the different vulnerabilities of the protocol

Blockchain technology is in its early stages and there are many technical challenges that need to be overcome before it become part of mainstream commercial technology. We have great confidence given the developer community that already exists around Ethereum as well as the valuable technologies it introduces. The primary power in this blockchain is the smart contract, which is able to function as a third party mediator, through code which cannot be manipulated. We were happy with our results in successfully deploying a realistic smart contract application on an Ethereum network, and being able to build Go API libraries on top of our application to make it user friendly. That being said, running computations on the real ethereum network is very expensive, and that is something that will have to be resolved before it becomes widely adopted.

Member Contributions

- Parth Shah and Vinit Adkar: Implemented the Blockchain Consensus Protocol described in 2 and performed the analysis described in 2.3
- Parth Shah: Implemented the visualization tool described in 2.4
- Amitai Porat and Avneesh Pratap: Implemented the Lockchain application over the ethereum Platform as described in 4

References

1. Douceur, J. R. The sybil attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, 251–260 (Springer-Verlag, London, UK, UK, 2002). URL http://dl.acm.org/citation.cfm?id=646334.687813.
2. Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system," http://bitcoin.org/bitcoin.pdf.
3. Proof of stake. https://en.bitcoin.it/wiki/Proof_of_Stake.
4. Hammerschmidt, C. Consensus in blockchain systems. URL https://medium.com/@chrshmmmr/consensus-in-blockchain-systems-in-short-691fc7d1fefe.
5. Ethereum design rationale. https://github.com/ethereum/wiki/wiki/Design-Rationale.
6. Ethereum introduction. URL http://ethdocs.org/en/latest/introduction/index.html.
7. Sompolinsky, Y. & Zohar, A. Secure high-rate transaction processing in bitcoin. In Böhme, R. & Okamoto, T. (eds.) *Financial Cryptography*, vol. 8975 of *Lecture Notes in Computer Science*, 507–527 (Springer, 2015). URL http://dblp.uni-trier.de/db/conf/fc/fc2015.html#SompolinskyZ15.
8. Buterin, V. Ethereum white paper. URL https://github.com/ethereum/wiki/wiki/White-Paper.