An Investigation into Stateless Blockchain

Yang Ji
Dept. of Computer Science
ID: 56064832

I. Introduction

Recent years have witnessed the prosperity of cryptocurrencies which seek for more efficient and secure financial transactions through removing trusted third parties like escrows and banks [1]. As their underlying technology, blockchain enables users to broadcast and confirm asynchronous transactions securely in an untrusted environment. In order to achieve this, nodes in blockchain systems need to validate broadcasted transactions by querying the current system *state*. To be more specific, this state can be available by storing subsequent ordered blocks from the genesis block in local.

However, due to the ever-growing nature of the blockchain systems, the storage requirement is increasing linearly with the ledger length. This restricted condition entails taking much storage space and time to download and maintain the entire history of blockchains. For example, a new full node that wants to join in the Ethereum network and acquires the correct ledger *state* needs to download 132.57 GB of data [2] currently.

As the emerging concerns on chain data size in the developer community, the concept of 'stateless' blockchain has received considerable interests nowadays. This concept is derived from a personal blog [3], which is proposed to avoid storing the whole state in an accumulative manner. Unfortunately, realizing a blockchain system with lessened storage requirement is a quite difficult task due to many challenges.

For one thing, maintaining a public ledger in a decentralized system entails the duties of validating and broadcasting transaction on each node. In order to check the validation of each transaction, nodes must store the current ledger *state* that determines the ownership of existing assets. For instance, the *state* of Bitcoin is a data set of unspent transaction outputs (UTXOs). In order to check the validation of transactions, nodes should confirm that spent coin belongs to UTXO set locally. In other words, nodes without storing UTXO set can't validate the incoming transactions.

For another thing, sharding or commitment schemes are viewed as effective methods to reduce the storage cost on each node. In these ways, nodes could verify transactions by checking the membership proofs provided by issuers. However, these solutions are actually at the cost of communication overhead across the system. And high communication overhead would inevitably lead to the low scalability and throughput of the entire network, which is against the principles of cryptocurrencies. How to leverage these tools to realize a cryptocurrency protocol is a challenging open question in blockchain community.

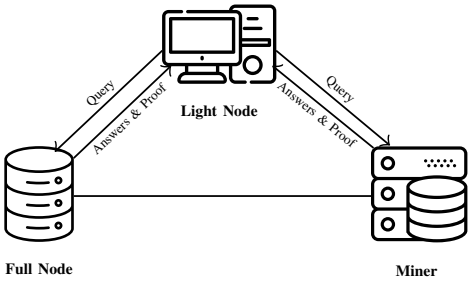

Fig. 1. System Architecture

With these unresolved problems, I conduct an investigation into state-of-the-art solutions of stateless blockchain in this paper. By categorizing existing literatures, I I then pick up some important literatures and explain their algorithms in more detail. By providing comprehensive evaluations for these methods,

The rest of this paper is organized as follows:

II. Application Setting

A. System Architecture

In this subsection, I will briefly introduce the system architecture of current blockchains. As shown in Figure, there exist three kinds of nodes with different functionalities in a classical blockchain network: *full nodes*, *miners* and *light nodes*. To fulfill the security model, full nodes must download blocks originated from the first block all the way and validate them against consensus proof. To be specific, a full node store all blocks including block headers and data records on the disk. Miners are full nodes with high computational power, who could execute consensus protocol (e.g. hash puzzle in Bitcoin) and generate new blocks for money rewards. Restricted in limited resources, a light node only needs to download block headers during the initial syncing process and then requests transactions from connected full nodes as needed.

B. Stateless Blockchain

In the current blockchain system, locally maintaining the validation state means quite cumbersome work. On one hand, this high storage requirement potentially fails those users who can't dedicate large storage space to join in the network. More than that, increasing storage requirement might also reduce the incentives for running full nodes, which could lead to possible centralization of blockchains. On the other hand, full nodes tend to leverage database (e.g. LevelDB in geth client) to

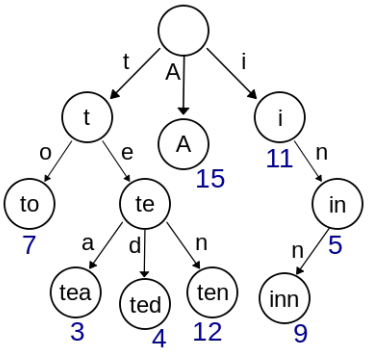

Fig. 2. Trie Data Structure

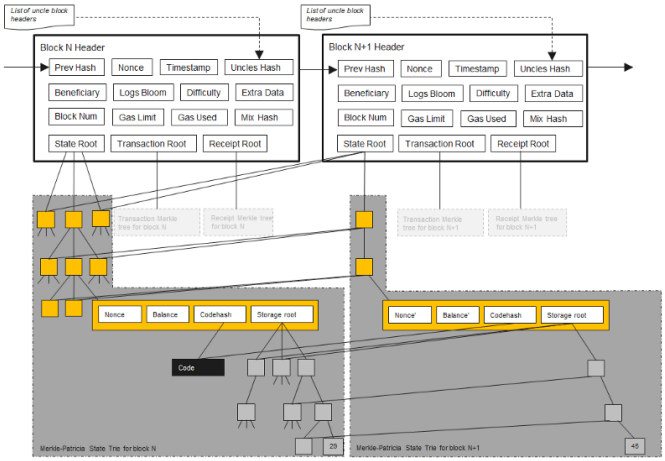

Fig. 3. Ethereum Storage Layer

manage the entire ledger *state*, which causes expensive I/O costs and prolong the validation time.

Due to the above concerns, the blockchain community is eager for a well-designed protocol to tackle these issues by lessening or even removing local validation state.

III. Background on Transaction Validation in Current Blockchain Systems

In this part, I will take Ethereum as an example to untangle the intricacies in the data storage layer that exists in full nodes. In Ethereum, developers have implemented many optimizations on the original data structure called Merkle Patricia Tries. These data in local databases and on-chain block headers jointly contribute to the 'World State' in blockchains. What's more, the concept of 'stateless' blockchain can be also applied into lightweight node. I will explain the details later.

A. Ethereum Data Storage Implementation

The core data structures which exist in Ethereum are made up of optimized Merkle Patricia Tries. The term trie is derived from the word 'Re**tri**eve' and the same as tree. Trie data structure, which is also called digital tree, radix tree and prefix tree, could be used to store the dynamic maps of key-value. We can efficiently traverse a branch of nodes in the trie with the reference to the corresponding keys. For example, in Figure 2, the sequence of English alphabet is used to comprise the key. In the path of retrieving the value, the same sequences are grouped together to avoid massive of wasted space.

Many optimized implementations have been applied into the Ethereum's trie data structure in order to increase the search efficiency and performance. In the Ethereum's official document, Merkle Patricia trie are defined as a cryptographically authenticated data structure that is used to store all bindings of key and value. The hash function of each node, as I have mentioned in my first assignment before, could guarantee the data integrity and provide fast verification for users. As for query processing, this unique data structure enables inserting, searching and deleting operations in the complexity of O(log(n)). In engineering, some encoding schemes like Hex-Prefix encoding and Recursive Length Prefix (RLP) are also used for improving the query performance. To conclude that, tries are enforced with cryptographic security by leveraging the hash function to reference each node. And these tries could provide the efficient inserting, searching and deleting operations in local databases (Leveldb or Rocksdb).

In order to maintain the state of the whole blockchain, Ethereum blocks contain four distinct trie data structures: (1)*Receipt Tree* (2)*State Tree* (3)*Transaction Tree* (4)*Storage Tree*. As illustrated in Figure 3, The hash roots of first three merkle patricia tries are stored in the block headers in the form of Keccak 256-bit hash value. The hash root of *Storage Tree* is included in *State Tree* with its RLP encoded data records.

In more detail, *Transaction Trie* is proposed to record transactions in the current block. Thus, once the block is appended into the main chain, *Transaction Trie* is also fixed and never updated. Same as *Transaction tree*, *Receipt Tree* is used to record the outcome of transactions and never updated. In Ethereum yellow paper [2], *Receipt Tree* could be used for zero-knowledge proofs or index and searches. *Receipt Tree* contains post-transaction state, cumulative gas used, logs and bloom filter created by the information of logs. Different from the above trees, *State Tree* records a key-value pair for each Ethereum account in an accumulative way. *State Tree* is actually only one global state tree which contains all past account information. Thus, each time a miner generates a new block successfully, she also needs to update this global state correspondingly. Because Ethereum leverages smart contract to achieve reliable computation in an untrusted environment, some extra space is needed to store these contract codes permanently. Ethereum introduces contract accounts to manage these contracts and each contract account has a separate storage trie for that.

IV. Research Directions of Stateless Blockchain

Guidelines: In this subsection, you should introduce the typical research directions for encrypted search, describe in detail the goals of each research direction in terms of functionality and/or security. You should also specify and discuss the design challenges in each direction based on your literature study and

or your personal perspectives. Besides, you should properly discuss the existing works and try to reflect the state-of-the-art. A possible organization structure is given below.

A. Basic Searchable Encryption

Goals. The description goes here.
Design challenges. The description goes here.
Related work. The description goes here.

B. Dynamic Searchable Encryption

Goals. The description goes here.
Design challenges. The description goes here.
Related work. The description goes here.

C. Boolean Searchable Encryption

Goals. The description goes here.
Design challenges. The description goes here.
Related work. The description goes here.

D. Any Other Possible Categories

Goals. The description goes here.
Design challenges. The description goes here.
Related work. The description goes here.

V. Conclusion

The conclusion goes here.

References

[1] D. X. Song, D. A. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. of IEEE Symposium on Security and Privacy*, 2000.

[2] "Ethereum storage requirement," https://etherscan.io/chart2/chaindatasizefast, accessed: 2019-04-04.

[3] "Delayed-txo-commitments," https://petertodd.org/2016/delayed-txo-commitments, accessed: 2019-04-04.