A Preliminary 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.

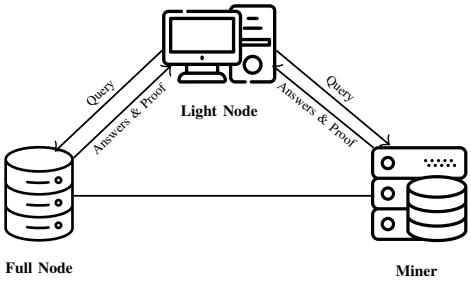

Fig. 1. System Architecture

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 provide the state-of-the-art solutions of stateless blockchain. I then pick up some important literatures and analyze their pros and cons in more detail. By providing comprehensive evaluations for these methods, I also provide some possible research directions on it.

The rest of this paper is organized as follows: the second paragraph provides a view of current blockchain system architecture and the history of stateless blockchain. Then I introduce the underlying mechanism of data storage and validating transactions in Ethereum in section 3. In section 4, I summarize the current studies on stateless blockchain. The section 5 gives an extension version of stateless blockchain in lightweight client. And the final section concludes the paper and lists some possible directions.

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)

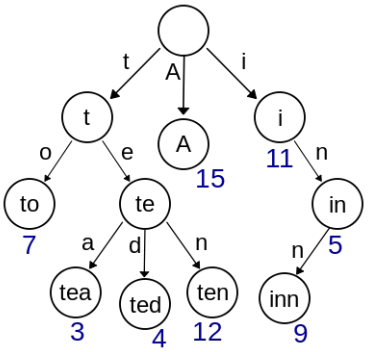

Fig. 2. Trie Data Structure

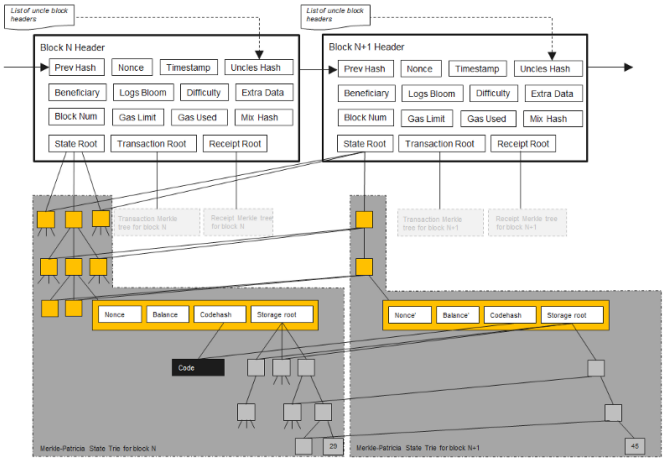

Fig. 3. Ethereum Storage Layer

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 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. Data Storage in Full Node

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 5, 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

System	Execution State	Proof Size	Bootstrap Security
Bitcoin [21]	UTXOs	Headers + TXs	Probabilistic (heaviest chain wins)
Ethereum [9]	All accounts	Headers + All accounts	Probabilistic (heaviest chain wins)
Permissioned	$\frac{\text{Live accounts}}{\text{Shards}}$	Majority of trust set's signatures	Cryptographic if majority never compromised; none otherwise
OmniLedger [17] + Chainiac [22]	$\frac{\text{UTXOs}}{\text{Shards}}$	$\frac{\text{Headers+Certificates}}{\text{Sparseness}} + \frac{\text{UTXOs}}{\text{Shards}}$	Cryptographic with static attacker; none with adaptive attacker
Algorand [14]	UTXOs	Headers + Certificates + TXs	Cryptographic
Vault	$\frac{\text{Live accounts}}{\text{Shards}}$	$\frac{\text{Headers+Certificates}}{\text{Sparseness}} + \frac{\text{Live accounts}}{\text{Shards}}$	Cryptographic

Fig. 4. Comparison of cryptocurrency models

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.

B. Data Storage in Lightweight Node

In practice, many devices with limited computing resources and storage space can't run as a full node. Therefore, in Bitcoin white paper [4], Satoshi Nakamoto puts forward a mechanism called simplified payment verification (SPV) to simplify transaction validation. Through utilizing this mechanism, lightweight client only needs to download all the block headers during the syncing process. Due to the lack of validation resources, lightweight node must query network nodes to download past blocks until she convinces that she has the correct ledger state.

In general, block headers contain enough information to assure that

1) Consensus proof (e.g. nonce) that is used to be against malicious provers.
2) specific transactions in blocks.
3) block depth used to convince the validation of transactions.

It is worth nothing that block depth is totally opposite to block height and corresponds to the cumulative difficulty originated from the genesis block. Once SPV client retrieves merkle branch and validates the existence of transaction in a block, she can only rely on the consensus mechanism for the security assurance.

IV. Research Directions of Stateless Blockchain

The concept of 'stateless blockchain' is derived from the Peter Todd's blog post [5], [6] because full nodes in blockchain might validate new appending transactions without storing all past blocks originated from the genesis block. Originally, nodes verify the correctness of new transactions by querying local databases that keep all state and history in blockchains. To address this issue radically, nodes could only need to store a short commitment to the current state instead of storing the whole ledger state. Thus, the issuer of transactions needs to add an assets proof for validation. In this way, individual nodes only need to store the a set of own UTXO/accounts along with their proofs. These proofs are used to prove the membership or non-membership of any elements in a set. The naïve accumulator is Merkle Hash Tree. There are also other authenticated data structures that have specific optimizations. Except for hash-based tree, vector commitment also provides the functionality of position binding. But these two methods have their own advantages and disadvantages. I will analyse and compare these two methods later. Since the concept of 'stateless blockchain' is quite new in related field, there are few literature studies on it. Hence, I will explain the state-of-the-art solutions comprehensively and list the trade-offs therein.

A. Hash-based Accumulator

Goals. The final purpose is to reduce the storage requirement for newly joining the network or validating the transactions. Since the protocol and mechanism of transaction validation has changed over, original authenticated data structure (Merkle Hash Tree in Bitcoin and Merkle Patricia tree in Ethereum) should be designed again.

Design challenges. Designing new authenticated index structures should consider the following aspects:

- **Short membership/non-membership proofs.** Since stateless blockchain needs an additional field (Merkle tree branch), overlong proof size might increase communication overhead inevitably. High communication overhead might lead to network latency and low the entire network throughput totally.
- **Support more efficient updating and inserting.** As a cryptocurrency, blockchain updates its global state frequently. Bitcoin generates a new block every $10 \sim 12$ minutes and in Ethereum generating a new block only needs $10 \sim 15$ seconds. Frequent updating operations on ledger state needs users to update local information to the newest.
- **Support any participant to update membership proofs.** As mentioned before, pending transactions need a uncertain time to be confirmed (Or never be confirmed due to low incentive or double-spend problem). However, the commitment of Merkle hash tree updates frequently, and as a result, the initial witness is not valid any more. To address this issue, we need to allow every miner to update witness freely.

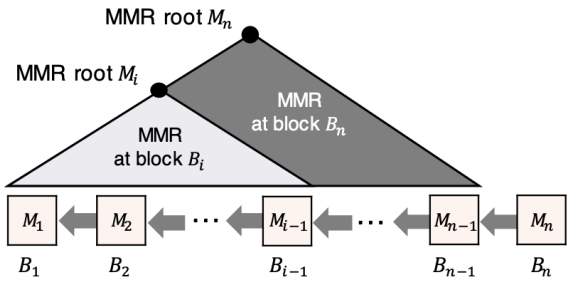

Fig. 5. Merkle Mountain Tree

Related work. Peter Todd proposed Merkle Mountain Tree (MMR) [5] used for distributed time-stamping services atop Bitcoin in 2013. The data structure of MMR allows individual nodes to append new data without reconstructing the tree from scratch and keep a balanced state all the time. As is shown in Figure 5, each time we append a subtree with new block headers, we only need to modify only a few nodes in the original tree and still keep the proof size in $\log n$. Another advantage of this authenticated data structure is that only a tiny modifications need be done on the base of current blockchain systems. However, this design lacks enough theoretical support including complexity analysis for updating and experimental result.

Based on MMR, Todd then put forward TXO (Transaction Output) commitment as opposed to UTXO commitment. Once an transaction output is appended into the TXO MMR, this output is never removed. To an extreme extent, individual nodes could only rely on TXO MMR, and as a result, eliminate the issue of unlimited growth of UTXO set. What's more, nodes only need to append new data instead of updating TXO set frequently. However, the complexity $\log n$ could increase bandwidth overhead inevitably. Hence, individual nodes also need to maintain UTXO set for recent transactions and TXO set is only needed when very old coins are spent. There are also some optimizations like storage allocation in cache and disk for UTXO and TXO sets. As for the generation of TXO/UTXO commitment proofs, anyone could produce and update these proofs without involving signers.

Even though, there are still some unresolved problems in this proposal as follows:

- **Incentive for storing less information.** Incentive mechanism in current blockchain systems actually requires miners to store all information in local database (even in RAM for high competitiveness). In my opinion, transforming all past UTXO history to an archived TXO MMR can't alleviate this issue at all since the growth of TXO set is also unlimited. If a miner forgets very old unspent transactions for saving storage, she might lose competitiveness against other miners. In other words, competitive nodes prefer to choose to download and maintain all history.
- **Increase burden of light node.** Including proofs in transactions requires lightweight nodes to request full nodes for help generate proofs.
- **High Communication Overhead.** Although TXO set is compressed and less likely to be used, its proof size could be very large in the complexity of $\log n$.

In Ethereum research forum, Justin Drake utilizes asynchronous accumulator [7] to construct witnesses. Asynchronous accumulators can provide position binding with low update frequency. The main benefits are that nodes only need to update witness logarithmically in the number of events. And, further on that, wwe can separate the history and state to construct 'dual accumulator' proof. As for immutable transaction history, we can utilize non-dynamic asynchronous accumulators for this large set. As for changing present transactions, we can still use current Merkle Patricia Tree to keep records. Here, an extreme approach is to leverage SNARKs/STARKs on smart contracts to generate efficient proofs for consumptions and non-consumptions.

Separating history and state might be an efficient way to realize stateless blockchain. However, many critical problems are unresolved in this proposal. This approach suffers from the problem of transaction synchronisation since miners might using this contract at the same time. Thus, accumulator sharding might be more suitable in this case. Same as the proposal of Todd, the witness size is still a big problem.

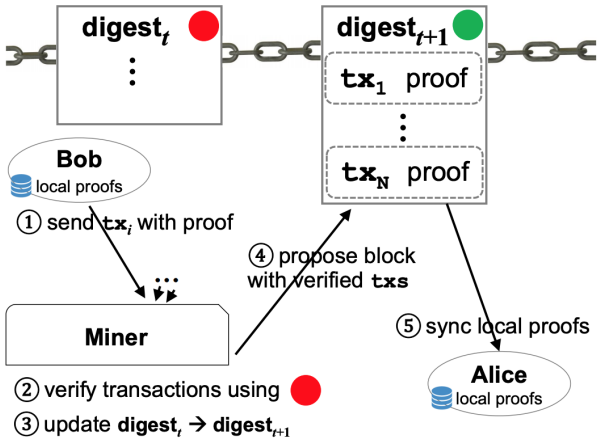

Fig. 6. Edrax Architecture

Based on prior discussions on the forums, several research studies on stateless blockchain have emerged in recent months. In Edrax [8], the authors utilizing the methods of verifiable data outsourcing (e.g. authenticated data structure [9], [10] and vector commitment [11]) to support stateless transaction validation. In this part, we focus on authenticated data structure used to support UTXO model. As shown in Figure 6, Edrax requires issuers of transactions to attach local proofs and clients to synchronize their local proofs for each new block.

In UTXO model, full nodes validate the new transactions by maintaining a set of unspent transaction outputs. Here, they utilize *sparse* Merkle Tree of 2^W leaves to represent UTXO set. In order to verify the correctness of the transactions, nodes need to check that input coins belong to this set. Operations involved in validating process include removing old coins and inserting new coins. The *i*-th leaf node would store the output of *i*-th transaction. And they set the value to be *null* to denote deletion of the coin. In hash-based accumulator, root hash could be used for validation digest and Merkle tree proofs are used as local proofs.

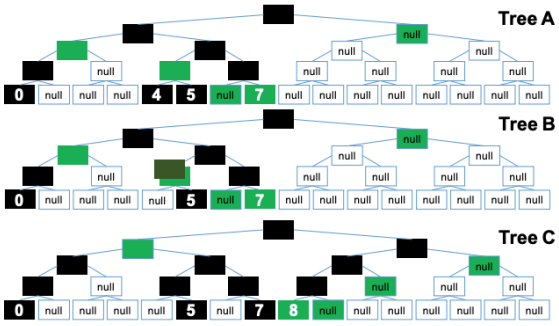

Fig. 7. Sparse Merkle Tree

In Figure 7, we represent a record in *sparse* Merkle Tree in the form of (i,[pk,v]), where i is an increasing counter number to label the output transaction, pk is a public key and the value of spent coins v. *Validation Digest* (the root hash of *sparse* Merkle Tree) is required to be appended into each block. And the local proof π for each transaction should be updated to the newest. As for clients, spending a transaction output (x,[pk,v]) needs to contain her public key PK and a local proof π for this transaction. As for miners, they need to check two constraints for a valid block. Firstly, they need to use public key PK to check the validity of its signature sig. Secondly, they need to ensure that all inputs of transactions are unspent by checking the local proof. Since transactions need uncertain time to be confirmed and new blocks can be generated during that time, we need to allow every miner to update this local proof without issuers. For simplification, we omit the detailed algorithm here.

Instead of enforcing full nodes to maintain the whole ledger, this method reduce the validation cost by requiring clients to maintain local proofs for own coins. Same as the methods before, the witness size is on the complexity of log$n$ where n is the number of all unspent coins in history. One optimization here is to use more efficient authenticated data structure such as authenticated red-black tree to replace *sparse* Merkle Tree. As for account-based model (e.g. Ethereum), the authors utilize vector commitment to provide position binding proofs.

There is little targeted optimization for hash-based accumulator in Edrax. Another work called Vault [12], recently published in NDSS, put forwards many special optimizations

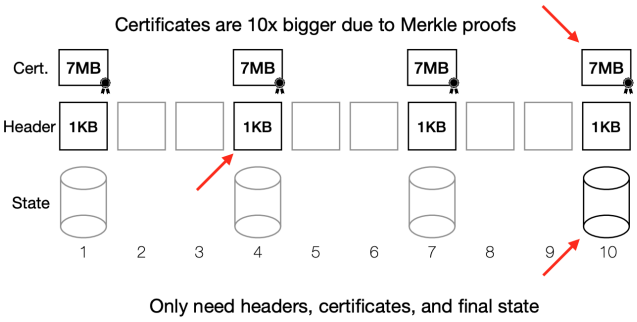

Fig. 8. Protocol for skipping blocks

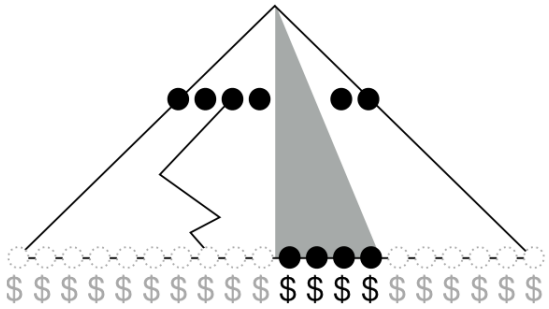

Fig. 9. An illustration of Vault sharding

for authenticated data structure and Algorand. In particular, to the problem of empty accounts in Ethereum, they put forward a scheme of transaction expiration. By setting the time window for each transaction, we don't need to track nonce of each account and can delete empty accounts safely. Enforcing transactions to expire means that, if a transaction can't be confirmed before it expires (e.g. might because few transaction fees), the issuer must resign this transaction again. For users, they can confirm whether their transactions are accept finally after the expiration.

Same as Edrax, vault also adopts *sparse* Merkle tree to store the account sets. Considering the communication overhead, vault requires nodes to locally store the merkle tree frontier which is an intermediate layer in Figure 9. Thus, transactions could only include subtree proofs that prove the Merkle Tree branch starting from the middle nodes instead of root nodes. It is in fact a trade-off between the storage and frontier depth.

B. Vector Commitment

Goals. Same as hash-based accumulator, vector commitment can also provide proofs for position binding.

Background. The approaches of Merkle tree can't be applied into account-based model directly because single local proof for one account can't update account information of sender and receiver. Considering the following case: Suppose that Alice has t_A coins and Bob has t_B coins. If Alice wants to transfer her 5 coins to Bob, she can just broadcast a

new transaction along with her local proof. To validate the correctness of her transaction, miners need to check whether her balance is larger than 5 ($t_A \geq 5$) using her local proof. Once miners confirm the validation of this transaction, they need to utilize this witness to update validation digest in the block header to reflect the balance change. Updating Alice's balance is very easy since Alice has attached her witness in the transaction. However, updating Bob's balance is impossible because miners don't maintain the global ledger state any more and transactions doesn't contain receiver's local proof. And involving Bob in the issuance of new transaction violates the design principle of cryptocurrency. To tackle this problem, Edrax leverages *algebraic vector commitment* to replace tree-based accumulator.

Design challenges. in this setting, a new vector commitment scheme need to be constructed considering the following perspectives:

- **Represent vectors in a succinct form.**
- **Update proof efficiently.**
- **Proof size and key size shouldn't be large.**

Related work. There are many literature studies on vector commitment. For verifiable computation, Yupeng Zhang et al. utilizes multilinear extension to represent vectors in a succinct manner. However, they don't present an efficient proof updating algorithm. Other vector commitment suitable to Edrax could be two-fold: One uses elliptic curve cryptography (ECC) to construct vector commitment. Another is based on RSA cryptography. Both of these methods suffer from the large key size or slow updating time.

In this paper, I will briefly introduce the secure vector commitment scheme in Edrax and key components in it.

Setup Phase. Here, given an upper bound n (the total number of accounts in Edrax) and a secure parameter λ, running $\texttt{KeyGen}(1^{\lambda},n)$ could generate prover key prk, verification key vrk and update key upk_i These security parameters could be hardcoded into the client side. To forbid the risk of trapdoor attacks, Edrax can employ multi-party computation to set up.

Validation Digest. In order to validate the transactions, each block should contain two values in the block header. The first is account digest $\texttt{digest}_t$. Just like the root hash of Merkle tree, $\texttt{digest}_t$ also records the balance of all accounts. The second is counter number of accounts $\texttt{cnt}_t$. This value represents the total number of accounts in Edrax.

Client Side. In client side, users need to create a new account and spend money. For creating a new account, user needs to generate a pair of public and private key firstly. Then she can issue a `Init` transaction to map her account into next number.

As for spending money, Edrax offers `Spend` transaction for users. In order to send this transaction, sender should contain her public key, receiver's public key, payment amount, local proof and sender's balance.

Miners As for the creation of new blocks, miner need to check three constraints:

- Check whether signature is corresponding to the public key pk
- Check this account could afford this payment amount
- Check the correctness of local proof

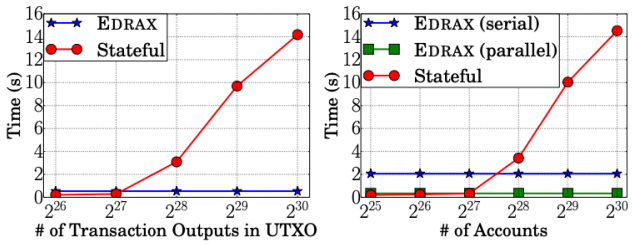

Fig. 10. Edrax vs stateful validation

After generating a new block, miners need to update the validation digest in the block header. Edrax provides `UpdateDigest` algorithm to update sender's and receiver's balance. Updating witness in block header means that participants in the network need to synchronize the validation digest. It actually increases the burden of lightweight clients and increase the communication overhead in the network.

Analysis Public key size is too large. In Edrax, since the upper number n is hardcoded into each client, we need to set n to be large enough. It is unknown that how clients get initial proof π during `Init` transaction in the stateless blockchain.

V. Extension of Stateless Blockchain

Stateless blockchain is proposed for reducing the data storage of miners. To further extend this concept, we even think of how to build a 'stateless' lightweight node. Currently, lightweight nodes need to store the block headers to validate the consensus proof. In general, the storage is linear with the total number of blocks. In Bitcoin, this is not a problem because new blocks are generated for 10∼15 minutes and the size of each block header is only 80 bytes. But Ethereum generates a new block only for 10∼12 second and the block header size is 508 bytes (much larger than ones in Bitcoin). SPV client in Ethereum needs to download 3.6 GB of data to confirm the consensus of the whole network. Thus, a reasonable stateless SPV client is need in Ethereum.

Kiayias et al. once proposed a method called non-interactive proof of proof-of-work (NPPoPoW). In this way, lightweight client only needs to store a polylogarithmic number of block headers. However, it suffers from the facts including ideal model (no adversaries) and only limited in proof-of-work system. In order to overcome above drawbacks, in this paper [13], the authors adopt merkle mountain tree commitment and propose a probabilistic protocol called flyclient. What's more, flyclient also has a smaller proof size of all parameters.

VI. Evaluation of Edrax

Authors compare the validation time of Edrax with stateful blockchains in UTXO and account-based model. Since Edrax sets an upper bound of accounts/UTXO at one time, validation time of Edrax is little worse than the stateful blockchains with small accounts. However, when the number of transactions

is increasing, validation time of stateful blockchains also increases linearly. Thanks to the attached proof, miners can still validate this transaction in a nearly constant time.

VII. Conclusion

As for stateless blockchain, both authenticated data structure and vector commitment are possible solutions. And both of these methods suffer from the witness size in the block header. In order to design an efficient and reasonable protocol, local proof synchronisation is a critical part. What's more, current studies can't support smart contracts in the stateless settings. Leveraging SNARKs in blockchain could help miners, even every node, validate transactions faster and achieve fast bootstrapping.

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.

[4] S. Nakamoto *et al.*, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[5] P. Todd, "Making utxo set growth irrelevant with low- latency delayed txo commitments," https://petertodd.org/2016/delayed-txo-commitments, accessed: 2019-04-15.

[6] P. Todd and OpenTimestamps, "Merkle mountain tree," https://github.com/opentimestamps/opentimestamps-server/blob/master/doc/merkle-mountain-range.md, accessed: 2019-04-15.

[7] L. Reyzin and S. Yakoubov, "Efficient asynchronous accumulators for distributed pki," in *International Conference on Security and Cryptography for Networks*. Springer, 2016, pp. 292–309.

[8] A. Chepurnoy, C. Papamanthou, and Y. Zhang, "Edrax: A cryptocurrency with stateless transaction validation," Cryptology ePrint Archive, Report 2018/968, Tech. Rep., 2018.

[9] C. Xu, Q. Chen, H. Hu, J. Xu, and X. Hei, "Authenticating aggregate queries over set-valued data with confidentiality," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 4, pp. 630–644, Apr. 2018.

[10] C. Xu, J. Xu, H. Hu, and M. H. Au, "When query authentication meets fine-grained access control: A zero-knowledge approach," in *Proceedings of the 2018 ACM SIGMOD International Conference on Management of Data*, Houston, TX, USA, Jun. 2018, pp. 147–162.

[11] D. Catalano and D. Fiore, "Vector commitments and their applications," in *International Workshop on Public Key Cryptography*. Springer, 2013, pp. 55–72.

[12] D. Leung, "Vault: Fast bootstrapping for cryptocurrencies," Ph.D. dissertation, Massachusetts Institute of Technology, 2018.

[13] L. Luu, B. Buenz, and M. Zamani, "Flyclient super light client for cryptocurrencies," accessed 2018-04-17.[Online]. Available: https://stanford2017 . . . , Tech. Rep.