TICK: Tiny Client for Blockchains

Wei Zhang*, Jiangshan Yu†, Qingqiang He* and Nan Guan *

*The Hong Kong Polytechnic University, China
†Monash University, Australia
‡Corresponding email: ss-wei.zhang@connect.polyu.hk

***Abstract*—Currently, a lightweight blockchain client cannot verify a transaction directly. Rather, it generally considers a transaction to be valid and finalized if there are six confirmations on this transaction. This is not desired for micro-payments where payees may opt for zero-confirmation.**

When a payee chooses to accept zero-confirmation transactions, it needs to verify the validity of the transaction. In particular, one of the steps is to verify that the input of the transaction is not previously spent. This requires the payee to either trust some full nodes to perform the verification, or download the entire blockchain in order to perform the verification locally. The former is less secure as anyone can be a full node, it cannot be blindly trusted; the latter is not desired since a blockchain is large in size and is ever growing.

We propose TICK, a Tiny Client for blockchains, to solve this problem. TICK advances blockchain client in the following aspects. First, TICK allows a mobile device to efficiently and cryptographically verify whether an input of a transaction is previously spent or not. Second, TICK can be adapted without changing the current implementation of the blockchain, such as Bitcoin. Rather, it only requires miners to put an extra hash value in the free field of a block. Last, as a side effect, rather than requiring new miners to download the entire blockchain to perform mining, the TICK design allows new miners to download only a small portion of data to start mining.

We implement TICK for Bitcoin, and provide an experimental evaluation on its performance by using the current Bitcoin blockchain data. Our result shows that the proof for verifying whether an input of a transaction is spent or not is only less than 2KB. The verification is very fast – a mobile client only needs to compute a few dozens of hash values. In addition, to start mining, new miners only need to download a few GB data, rather than downloading over 200 GB data.

I. Introduction

Since the start of Bitcoin [1], especially since its fast growth in 2013 and 2014, blockchain has emerged as a very attractive technology that promises tremendous potential for creating new applications and business models. One of the most exciting aspects of blockchain technology is that it is entirely decentralized, rather than controlled by one central point [2]. The lack of a single authority makes the system fairer and considerably securer. Although Bitcoin is the most successful application of blockchain, blockchain technology has potential in a wide range of application areas beyond cryptocurrencies, such as finance [3], healthcare [4], reputation system [5].

As a decentralized system, all the nodes in the system collect data such as transactions from the underlying peer-to-peer network, verify their correctness and group them as a block. Then the generated block is broadcast to other nodes by a gossip protocol, and eventually all the nodes maintain all the transaction history locally and agree on a unique order in these transactions. As a result, it imposes a heavy requirement on the storage capacity of blockchain nodes. For example, the size of transaction history of Bitcoin and Ethereum are 200 GB and 554.67 GB, respectively. Such a huge storage requirement makes applying blockchains on storage-limited devices (e.g., mobile devices) almost impossible. With consideration of the indispensability of mobile payment in our daily life, supporting mobile payment is essential for the development and popularity of blockchains. Therefore, the storage requirement should be reduced in order to adapt to the mobile devices.

To break the barriers caused by the heavy storage requirements, instead of downloading the whole transaction history, most blockchains support the lightweight client, also known as the light client. The light client only needs to download the block headers and use the simplified payment verification (SPV) to verify transactions. On the surface, the light client is more applicable for mobile devices, but it still has several limitations. On one hand, although only maintaining block headers can significantly reduce the storage requirement, its data size still increases linearly with the number of blocks without any limitations. Therefore, with the growth of the blockchain height, there is also a risk of exceeding the storage capacity of mobile devices. On the other hand, currently, when users issuing transactions, users have to wait for at least 6 confirmations (i.e. the block containing the transactions is followed by at least 5 blocks in the chain), before considering your transaction is permanently recorded. However, in many cases, a payee cannot wait for an hour to confirm a transaction, and therefore the zero-confirmation transaction [6] is proposed and considered by many Bitcoin merchants. That is, merchants may consider to accept micro transactions with no confirmation in the blockchain, as this provides a faster way to manage small-value transactions. However, since no miner has verified this transaction yet, it is vital for the merchants to at least verify the validity of the transaction. Unfortunately, the light client can not validate transactions as it only maintains all the block headers, and thus it can not perform such fast payment for micro transactions.

Miners play an important role in the proof-of-work (POW) based blockchains, they compete for the right to generate the new block and then get a reward. During the competition, miners validate transactions and audit the whole blockchain, which can guarantee the system's security. In principal, if an attacker can control more than 51 percent computing power, it can control the system. Therefore, more miners in the system, controlling more than 51 percent computing power is more difficult and the system is more stable. However, in existing blockchains, only a full node who maintains all the transaction history can be a miner. As mentioned above, a full node needs to download the full transaction history in advance. However, downloading such a huge amount of data is time and storage consuming which may generally take several days. Consequently, the system may lose some potential miners, which is a big loss for the system.

In this paper we propose TICK, a Tiny Client for blockchains. In TICK, we add an ordered data structure [7], [8], called as UTXO tree, to maintain all the unspent coins. The unspent coin, also known as Unspent Transaction Output UTXO in blockchains, is the only sources of input for the following transactions [9]. With the UTXO tree, instead of checking the transaction history, transaction verification can be implemented by checking the presence or absence of all inputs in the UTXO tree, in the size of O(log N). Consequently, a light client only needs to download a fixed number of block headers rather than all the block headers, and thus the storage requirement of a light client is constant and very small, make it suitable for mobile devices. In addition, the proposed light client can validate transactions through request proofs to the full node, and thus make it possible for mobile devices to implement fast payment with zero-confirmation micro transactions. In TICK, a miner can start to mine while only downloading a fixed number of blocks and the UTXO tree. Compared with the conventional system, the miner downloads much less data without sacrificing its functionality. Our contribution can be summarized as follows:

- We propose TICK, the first blockchain system that gives the ability of transaction verification to light client. Zero-confirmation fast payment, which is essential for blockchains, needs the ability to validate transactions. In contrast with existing light client, the proposed light client can implement fast payments without any third party.
- In TICK, a light client only needs to download a constant and small size of data. Mobile devices generally have low storage capacity, therefore the applications for mobile devices should cost less storage space and the cost should be relatively constant. Different from the existing light client, the proposed light client maintains a constant low size of data which is more suitable for mobile devices.
- TICK reduces the data size that a miner needs to download before mining. In conventional blockchains, a miner needs to download the whole transaction history which may cost several days and consume hundreds of GB memory. But in TICK, a miner can start to mine while only downloads several GB of data.

Our work is applicable for all UTXO-based blockchains. However, for simplicity, we demonstrate how it works by using Bitcoin as an example. We collect the Bitcoin data since its genesis block created in 2009 to date (29th March 2019). Experimental results show that, in TICK a light client only needs to download 672 bytes data which is tiny in contrast with the 43MB data of Bitcoin, and the data size is constant rather than that of Bitcoin which is linearly increased. Moreover, the light client in TICK can validate transactions through requesting proof from the full node. In past two years, the size of proof is less than 2KB which is small and constant. In TICK, different from Bitcoin which needs user downloading around 200GB data before starting to mine, a miner can start to miner while only downloads around 2GB data. As a result, the time cost and storage cost are both smaller than Bitcoin.

II. Related Work

In order to reduce the blockchain storage overhead, block summarization are proposed in several works [10], [11]. The main idea behind the block summary is that instead of storing all the blocks, nodes can store the change in a sequence of blocks (called block summary). This summary can store all the input resources for the given blocks and the total change that was introduced by these blocks. But the compression ratio of the block summarization method not high enough for mobile devices. For example, in Bitcoin, they achieve a compression ratio of 0.54, as a result, a node still needs to download around 100 GB data.

Kiayias et al. proposed a sublinear light client [12], called proofs of proof-of-work (PoPoW), which allows light client to download a logarithmic number of blocks. They also proposed a non-interactive PoPoW (NIPoPoW) protocol [13] that allows succinct proofs but is with the same proof complexity as in [12]. However, both these two methods only work if a fixed difficulty is assumed for all blocks. However, such an assumption is infeasible for most proof-of-work based cryptocurrency blockchain systems. For example, in Bitcoin, the block difficulty exponentially increases over the network's lifetime in the past decade [14].

Flyclient [15] is a concurrent work proposing a light client for blockchain. In their work, the light client also only needs to download a logarithmic number of block headers to synchronize the blockchain. However, users in Flyclient cannot verify the validity of transactions without downloading all transactions of size O(N).

There are some works that protect the privacy of light client which are orthogonal to our method. In order to verify transactions, a light client needs to request data from full node through a peer to peer network. However, such a payment verification may leak considerable information about the clients, thus defeating user privacy that is considered one of the main goals of decentralized cryptocurrencies. In work [16], [17], they use available trusted execution capabilities, SGX enclaves and Trusted Execution Environment to protect the user privacy.

III. Preliminaries

Using Bitcoin as an example, we briefly introduce the UTXO based blockchains in Section III-A. Then we show its limitations in Section III-B and discuss the reasons for these limitations in Section III-C.

A. Bitcoin

Bitcoin [1] is the first and still the most popular cryptocurrency system. Using Bitcoin, users can issue transactions without any powerful central authorities. Bitcoin includes two types of transactions: the coinbase transaction and the standard transaction. Coinbase transactions, the first transaction of each block, create new coins from nothing by the miner. Standard transactions, the most common one, record the coin transfer between the payer and the payee. In Bitcoin, an account is in essence a public-/private-key pair. We show the structure of a standard transaction in Fig. 1. A standard transaction contains a Txid, at least one input and at least one output, where Txid is the unique ID of the transaction and it can be used to uniquely identify the transaction. An output contains value which denotes the coins received by the payee, and the scriptPubKey which is the public key of the payee. Therefore, when the transaction is recorded on the blockchain permanently, all the nodes agree that the payee receives value coins through the transaction. An input contains Txid, vout and signature, where signature is singed by the payer with its private key and used to make sure the transaction is issued by the payer himself. The combination of Txid and vout can be used to uniquely identify a previous output belonged to the payer. Therefore, when the transaction is recorded on the blockchain permanently, all the nodes agree that the referred output is consumed by the payer in the transaction. For example, if we have the following input:

$$(\mathsf{Tx}20, 3, \mathsf{signature})$$

then the referred output is the output3 of Tx20. Therefore, the coin received in output3 of Tx20 is consumed, and it can not be referred by other inputs. A coinbase transaction is only slightly different from standard transaction data. The main difference is its single ”blank” input. Therefore, we do not introduce the coinbase transaction in detail.

Transactions are included into blocks by miners and then hashed as parts of a Merkle tree. Before grouping transactions into block, miners should verify them and make sure each of them is valid. This process is also known as the famous mining process. In a transaction, each input refers outputs of previous blocks to indicate the payed coins in the transaction, meanwhile the referred output can not be referred by other inputs. The output that does not referred by any input is also called as Unspent Transaction Output (UTXO). Therefore, when a new transaction is issued, the major job of transaction verification is to make sure the referred outputs are UTXO. In Bitcoin, the transaction verification is implemented through the following four steps.

1) The transaction is issued by the payer.

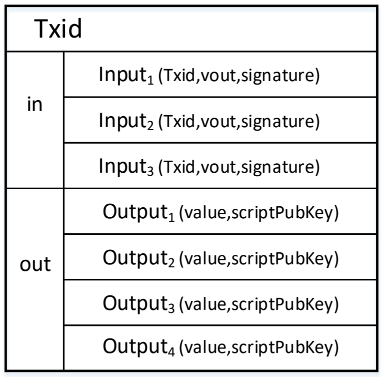
Txid
in
Input$_1$ (Txid,vout,signature)
Input$_2$ (Txid,vout,signature)
Input$_3$ (Txid,vout,signature)
out
Output$_1$ (value,scriptPubKey)
Output$_2$ (value,scriptPubKey)
Output$_3$ (value,scriptPubKey)
Output$_4$ (value,scriptPubKey)

Fig. 1: A standard transaction

2) In each transaction, the sum of value of all the outputs is no larger than the sum of value of all the referred outputs.
3) The referred output is existing.
4) The referred output does not referred by other inputs.

Among these four steps, the first one can be easily performed by checking the signature of each input. The completion of the last three steps needs to check the transaction history. Specifically, the 2nd step needs to check wether all the referred outputs are exist in previous transactions or not; the 3rd step needs to compute the sum of values of all the referred outputs; the 4th step needs to make sure that all the referred outputs are not referred by any inputs of the following transactions.

Since the mining process is not coordinated by any central party, different miners may generate different blocks at the same time and the new generated block may be added after the same block. Therefore the blockchain may fork into multiple chains. To agree on the same chain consistently with other nodes, each node downloads all blocks in every chain and picks the one with the highest total difficulty to follow. Using this strategy, it is shown that, in the long run, the network will agree on a single chain [18], [19], [20], known as the main (valid) chain. In Bitcoin, a block is considered on the block permanently if there are 5 blocks behind it. Also a transaction is agreed by all the nodes after 6 confirmations.

In a block, transactions are stored in a Merkle tree. We show a template Merkle tree in Figure 2. In order to differ from another Merkle tree defined in the following, we call the Merkle tree that stores transactions as **transaction tree**. In the transaction tree, the leaf nodes are transactions and the non-leaf nodes are the hash of its children nodes. The hash of the tree, root hash, is stored in the block header. Then the block header is hashed as Prev Hash, which is stored in the next block header. So that, any modifications to the transaction history will lead to a different root hash value and different Prev Hash in the next block. So any modification to the transaction history can be detected by comparing the computed root hash with the root hash stored in the block header. The Prev Hash is also considered as the pointer that connects the block with its previous block, and through the pointer different blocks are connected together as a blockchain.

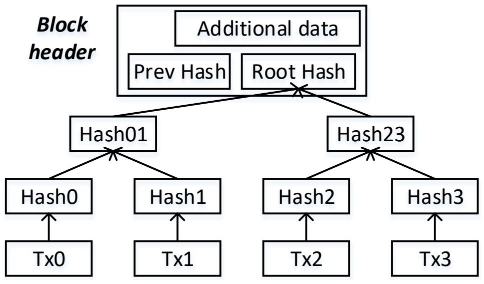

Fig. 2: A Merkel tree

In Bitcoin, full node is the one that stores all the blocks and it should has a huge storage space. Obviously, the full node can verify transactions as it contains all the transaction history. However, the data size of the whole transaction history is huge, and it is hard to be deployed on the storage-limited mobile devices. In order to reduce the storage requirement, Bitcoin also supports the light client which only stores the headers of each block. However, the light client can not verify transactions, while it can only use the Simplified Payment Verification (SPV) protocol to perform the first 3 steps of transaction verifications. For the example shown in Fig. 2, if a light client wants to know whether $\mathsf{Tx}0$ is in the block, it requests $\mathsf{Tx}0$ verification to a full node, which in turn replies hash values of $\mathsf{Hash}1$ and $\mathsf{Hash}23$. These return hash values, also called as critical hashes of $\mathsf{Tx}0$, are sufficient for the light client to reconstruct the merkel tree and compute the root hash. If the computed root hash equals to the root hash stored in the block header, the light client is convinced that $\mathsf{Tx}0$ is in the block. Therefore, it can verify an output is in $\mathsf{Tx}0$ (step 3) and get value of it (step 2), but it can not verify whether the output is referred by any inputs of following transaction (step 4).

B. Limitations

Mobile payment is indispensable in our daily life, and supporting mobile payment is important for blockchains. In order to deploy the blockchain on mobile devices, light client which only stores each block's block header is proposed. Compared with the full node, although it has a significantly less storage requirement, it also has some limitations. On one hand, due to the lack of information on transaction history, it can not validate transactions. As discussed above, through the SPV protocol, a light client can validate that a transaction is in a particular block, but it can't validate that it hasn't been redeemed by a subsequent block. More specifically, it can not verify whether an output is a UTXO or not. But the transaction verification is essential for fast payment. Unfortunately, the transaction verification in light client is insufficient and it cannot perform the fast payment. On the other hand, although light client needs less the storage space compared with full node, the required storage space still increases linearly with the number of blocks. For example, the Ethereum blockchain has more than 7.4 million blocks [21] and each block header is 528 bytes. Consequently, a light client in Ethereum should store more than 3.7 GB of data and the size increases 528 bytes every 13 seconds. Therefore, with the development of the blockchain system, the number of blocks increases fast and there exists a risk of that the data size of light client may exceed the storage capacity of mobile devices. In summary, when applying the light client on mobile devices, there exist the following limitation.

Limitation 1. *The* light client *can not validate transactions, so that* light client *can not perform fast payment. Moreover, its data size increases linearly and unlimitedly with the number of blocks, which is infeasible for the storage-limited mobile devices.*

Another limitation of the blockchain system is that it only allows the full node participating in the mining process. However, becoming a full node requires the user downloading the full transaction history which is time-consuming and storage space consuming. For example, the size of the whole transaction history in Bitcoin and Ethereum is more than 200 and 2.2TB [22], [21] respectively. Such a huge amount of data is infeasible for storage-limited devices and may consume a huge amount of memory space. More importantly, even a user with enough storage capacity also needs several days to synchronize its local blockchain, and then cause a long delay. Note that, Mining is an important part of the blockchain system which ensures fairness and keep the blockchain network stable and secure. More miners in the blockchain system, more stable and more secure the system is. However, such a long delay and a high storage requirement may makes the system losing some potential miners, which is big harm to the system's security and stability. To summarize, such a huge amount of data that a miner needs to store may bring the following limitation.

Limitation 2. *A* miner *must download the full transaction history, which is time-consuming and storage space consuming. Consequently, it not only makes the storage-limited devices impossible to be a miner, but also may lose many potential miners due to its high storage requirement and long synchronization time.*

C. Discussion

The root cause of **Limitation** 1 and **Limitation** 2 is that the size of data (full transaction history) for the transaction verification is too large. The miner needs to verify transactions, so that it must store all the transaction history. The light client needs to check whether a transaction is in the block or not, so that it must store all the block header, and even so it still can not validate transactions. However, as mentioned in Section III-A, not all the information in the transaction history is useful for transaction verifications. Among all the outputs of the transactions recorded on the blockchain, only the output that is a UTXO can be used for the following transaction. Specifically, if a referred output is a UTXO, it must exist in previous blocks (step 3) and must not be spent (step 4) yet, and through check the unspent output we can get its value (step 2). So, for transaction verification, these outputs that are not UTXO are redundant and do not need to maintain. Therefore, to verify transactions, we only need to maintain all the UTXOs.

IV. Methodology

As discussed in Section III-B, conventional blockchains have many limitations due to its huge size of the transaction history. In this section, we propose TICK to break these limitations. In contrast with the conventional light client which data size linearly increases over time, the data size of a light client in TICK is significantly less and constant. Consequently, the proposed light client is more applicable to mobile devices. In addition, the proposed light client can verify transactions, and thus the light client is given the ability to perform fast payment. In TICK, different from the conventional blockchains, the miner does not need to download all the transaction history, and thus the storage requirement of the miner is significantly less. As a result, miners can save a huge amount of storage space and plenty of time.

A. Overview

In this section, we present the overview of TICK. In TICK we introduce a new data structure to maintain all the UTXOs and a new hash value which is 32 bytes in the block header. TICK can be adapted without changing the current implementation of the blockchains, and it only requires users to put the added data in the free field.

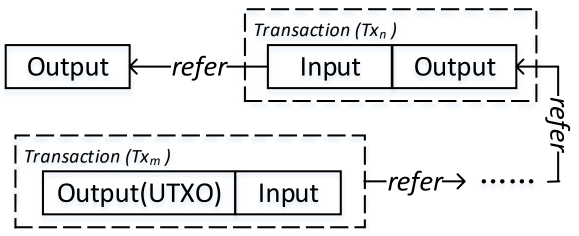

Fig. 3: UTXO in transaction history

As discussed in Section III-B, some transaction history is redundant for transaction verification. Among all outputs in the transaction history, only outputs that are UTXOs are necessary. More importantly, in the transaction history, only a small part of outputs are UTXOs [23]. Therefore, instead of downloading all the transaction history, a node can verify transactions if it only stores all the UTXOs. For example, backward traversing an output, the reference list among outputs and inputs can be obtained as shown in Fig. 3. In the list, only the latest output (output in Tx_m) is a UTXO, and all the past transactions are redundant for the transaction verification. Therefore, in this paper, we build a tree to maintain all the UTXOs for the transaction verification.

We denote the tree that stores all the UTXOs as UTXO tree. It is an ordered Merkel tree, and its leaves are lexicographically ordered UTXOs. The UTXO tree allows one to insert, delete and modify its data. Note that, maintaining the UTXO tree does not need to modify the block structure of the conventional blockchains. Then the root hash of the UTXO tree is stored in the block header as a part of it. When adapting TICK in existing blockchains, the root hash of the UTXO tree can be stored in the free field of each block. Therefore, any malicious modifications to the UTXO tree will lead to a different Prev Hash in the next block and will be detected. Since the UTXO tree is an ordered data structure, the user can efficiently check whether an output is in it or not. If an output is in the UTXO tree it is a UTXO, otherwise it is not a UTXO. Therefore, a node can verify transactions with the UTXO tree rather than all the transaction history. The overview structure of the proposed blockchains is shown in Fig. 4. We do not maintain UTXO trees for each block, on the contrary, at a point, there is only one UTXO tree stored in the full node. The UTXO tree records all the UTXOs at the point of the birth of the latest block. When the new block is generated, we can get the new UTXO tree through function Update, also the UTXO tree of past blocks can be obtained through function Rollback, where these functions are detailed in Section IV-B.

With the UTXO tree, a miner does not need to download all the transaction history and it only needs to download a preferred small number of recent blocks and the UTXO tree from a full node. Since the UTXO tree maintains all the UTXOs, the miner can verify transactions and starts to mine. As a result, the required storage space of the miner is significantly less than that of a full node. Similarly with the miner, since all the UTXOs are summarized and its digest is stored in each block header. The light client does not need to store all the block headers to verify transactions, and it only needs to download a preferred small number of recent block headers from a full node. Denoting the number of block headers or blocks that a light client or a miner wants to download as k, the user can determine the value of k according to its storage capacity, but the user needs to make sure that at least one block in the main chain is downloaded. For example, in Bitcoin, k can not be less than 6.

B. Full node

A key component of TICK is the UTXO tree which stores all the UTXOs in lexicographic order. In TICK, the transaction verification of miners and light clients relies on the UTXO tree. Therefore, compared with conventional blockchains which use all the transaction history to verify transactions, TICK can significantly reduce the storage requirement of miners and light clients. When the current blockchains, i.e., Bitcoin adapt TICK, it only needs to extract all the UTXOs from the whole transaction history and build the UTXO tree.

The UTXO tree is an ordered Merkel tree. Its leaves are UTXOs, and all the leaves are ordered depending on their hash value. The value of each non-leaf node is the hash value of its children nodes. For example, we show a template UTXO tree $\mathsf{T}_{\mathsf{UTXO}}$ in Fig. 5. $\mathsf{T}_{\mathsf{UTXO}}$ contains 4 UTXOs: UTXO_1, UTXO_2, UTXO_3 and UTXO_4. They are sorted depending on their hash value: $\mathsf{Hash}(1) < \mathsf{Hash}(2) < \mathsf{Hash}(3) < \mathsf{Hash}(4)$. Note that, the UTXOs can also be sorted with other values, i.e., issue date, and that can be determined by the system designer. In this paper, we use the hash value to illustrate our method.

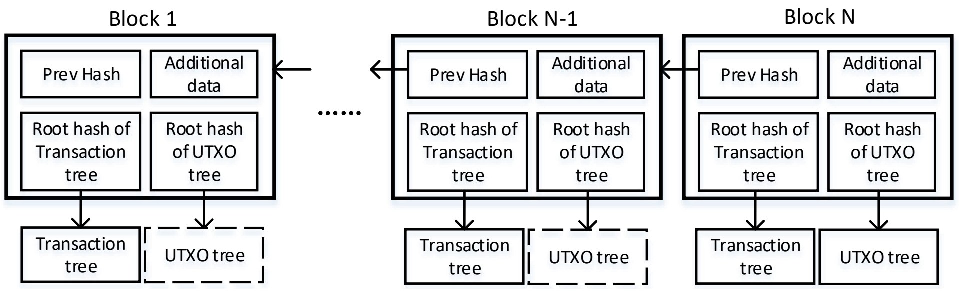

Fig. 4: Overview of TICK

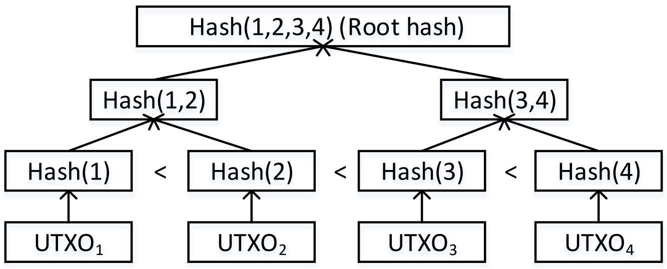

Fig. 5: UTXO tree $\mathsf{T}_{\mathsf{UTXO}}$

The hash value of the root is stored in the block header as the digest of the UTXO tree. Since the UTXO tree is ordered lexicographically, it supports efficient proof that some data is absent from the tree. In addition, as a Merkel tree, the UTXO tree supports efficient proofs that they contain specific data. Both proofs of absence and presence require an amount of data proportional to the logarithm number of UTXOs in the tree. Table I shows some methods that a UTXO tree supports.

TABLE I: The methods supported by $\mathsf{T}_{\mathsf{UTXO}}$.

Method	Input	Output
Size	$\mathsf{T}_{\mathsf{UTXO}}$	Number of UTXOs in $\mathsf{T}_{\mathsf{UTXO}}$
Root	$\mathsf{T}_{\mathsf{UTXO}}$	The root value of the UTXO tree $\mathsf{T}_{\mathsf{UTXO}}$
PoP	($\mathsf{T}_{\mathsf{UTXO}}$,UTXO)	Proof of Presence: The proof that UTXO is in the UTXO tree $\mathsf{T}_{\mathsf{UTXO}}$.
PoA	($\mathsf{T}_{\mathsf{UTXO}}$,UTXO)	Proof of Absence: The proof that any UTXO is absent from the UTXO tree $\mathsf{T}_{\mathsf{UTXO}}$.

We give some examples based on Figure 5 to show how the proof can be done with a UTXO tree $\mathsf{T}_{\mathsf{UTXO}}$.

Example of *proof of presence* To prove that a particular output is a UTXO, the full node needs to prove that the output is in $\mathsf{T}_{\mathsf{UTXO}}$. Then the full node needs to provide the critical hashes of UTXO_1 to compute the root of the tree, the path of the output in $\mathsf{T}_{\mathsf{UTXO}}$ and the value of the output.

$$\mathsf{PoP}(\mathsf{T}_{\mathsf{UTXO}}, \mathsf{UTXO}_1) = (\omega, \mathsf{Hash}(2), \mathsf{Hash}(3,4), \mathsf{value})$$

where $\omega = l, l$ represents the path from the root to UTXO_1, and l (resp. r) indicates the path to the left (resp. right) child. value represents the value of the output. So, given UTXO_1 and the proof $\mathsf{PoP}(\mathsf{T}_{\mathsf{UTXO}}, \mathsf{UTXO}_1)$, one can verify the proof by re-constructing the root value $H_T = \mathsf{H}(\mathsf{H}(\mathsf{H}(1), \mathsf{H}(2)), \mathsf{H}(3,4))$, where $\mathsf{H}()$ is the hash function that returns the hash value of inputs. If the computed root hash equals to the root hash stored in the block header, the proof is valid.

Example of *proof of absence* To prove that a particular output is not a UTXO, the full node needs to prove that the output is absent from $\mathsf{T}_{\mathsf{UTXO}}$ by performing the following steps.

- Locate node UTXO_i such that its hash value is lexicographically the largest one smaller than output.
- Locate node UTXO_j such that its hash value is lexicographically the smallest one greater than output.
- Prove that UTXO_i and UTXO_j are present in $\mathsf{T}_{\mathsf{UTXO}}$, and they are siblings (so no node is placed in between of them). The former is proved by using proof of presence of UTXO_i and UTXO_j, and the latter one can be verified by checking the path to UTXO_i and UTXO_j. Therefore:

$$\mathsf{PoA}(\mathsf{T}_{\mathsf{UTXO}},\mathsf{UTXO}_\mathsf{x}) = (\mathsf{PoP}(\mathsf{T}_{\mathsf{UTXO}}, \mathsf{UTXO}_\mathsf{i}), \mathsf{PoP}(\mathsf{T}_{\mathsf{UTXO}}, \mathsf{UTXO}_\mathsf{j}))$$

In the system, a full node only maintains one UTXO tree of the latest block. It records all the UTXOs when the latest block is added on the chain. When a new block is generated, the UTXO tree can be updated through the Update function defined as follows.

ALGORITHM 1: Update($\mathsf{T}_{\mathsf{UTXO}}$, *block*)

```
1:  for every Tx in $block$ do
2:      for every input in Tx do
3:          delete the referred UTXO from $\mathsf{T}_{\mathsf{UTXO}}$
4:      end for
5:      for every output in Tx do
6:          add the output into the proper position of $\mathsf{T}_{\mathsf{UTXO}}$
7:          according to its hash value.
8:      end for
9:  end for
10: Re-construct $\mathsf{T}_{\mathsf{UTXO}}$.
```

The Update function is performed with each transaction Tx in the new block. For each Tx, the Update function first deletes all the outputs referred by Tx from T_{UTXO}, and then adds all the new outputs introduced by Tx to the T_{UTXO}. After examining all the Tx in the new block, it updates the corresponding hash value and gets the root hash of the new T_{UTXO}. The root hash is then stored in the generated block header. A full node only contains one UTXO tree which maintains all the UTXOs at the latest blocks. However, there may be some splits in the chain, the latest block is not definitely on the main chain eventually. In order to track the main chain and get the T_{UTXO} on other splits, we define the Rollback function in the follows to get the T_{UTXO} of previous blocks. Similarly with the Update function, the Rollback function just does the opposite. Combined with the Update function, T_{UTXO} of different blocks at different splits can be obtained.

ALGORITHM 2: Rollback(T_{UTXO}, *block*)

```
1: for every Tx in the $block$ do
2:   for every output in Tx do
3:     delete the corresponding UTXO from $T_{UTXO}$.
4:   end for
5:   for every input in the Tx do
6:     add the referred output into the proper position of $T_{UTXO}$.
7:   end for
8: end for
9: Re-construct $T_{UTXO}$.
```

Compared with the full node in the conventional blockchain system, the storage overhead is only the one hash value (i.e., the root hash of the UTXO tree) which is 32 bytes for each block and the UTXO tree. For example, by now, the size of the UTXO tree in Bitcoin is around 4 GB which is significantly less than its original size 200 *GB* [22]. Generally, a full node has large storage capacity and such a storage overhead is relatively slight for full node.

C. Miner

In conventional blockchains, only the full node can participate in the mining process, because the transaction verification needs the whole transaction history. Therefore, a node needs to download a huge amount of data which is time-consuming and storage consuming. In addition, such a huge storage requirement makes performing the mining process on storage-limited devices near impossible. Consequently, the system may lose some potential miners and lose the chance to enhance the security of the system. In TICK, except to be a full node, a miner has an additional option. It only needs to download a part of the transaction history and then can start the mining process. And the size of data that a miner needs to download is significantly less than the data size of a full node.

The structure of the proposed miner is shown in Fig. 6. A miner needs to download a UTXO tree and a preferred number of recent blocks from the full nodes. In order to track the main chain, among all the stored blocks, there must exist

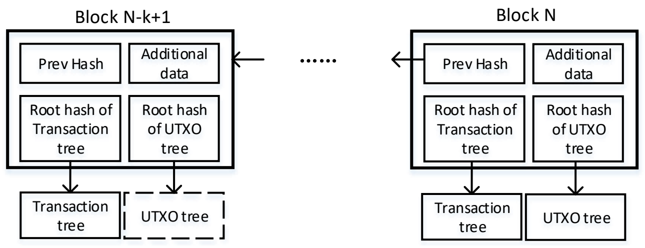

Fig. 6: Structure of miner

at least one block on the main chain. In the example, the miner downloads k blocks and a UTXO tree of the latest block Block N. The value of k can be determined by the miner according to its storage capacity. When a new block is generated on the longest chain, in principal the oldest block on the main chain can be deleted. Therefore, the size of data that a miner needs to maintain is relatively constant.

In TICK, the mining process includes the following 4 steps.

a Solve a puzzle.
b Generate the hash value of the previous block header.
c Verify all the transactions that they want to group in the new block.
d Build the UTXO tree for the new block.

Among these 4 steps, the combination of the first 3 steps is the mining process of the conventional POW-based blockchains. Step d is used to update all the UTXOs and build the UTXO tree for the new block. Since the miner stores the UTXO tree and the latest block, it can verify transactions (step c), generate the hash value of the latest block header (step b) and update the UTXO tree (step d). Apparently, step a can also be implemented by the proposed miner. Therefore the miners do not lose any functionality, as it can still validate transactions, contribute computing power to the system and ensure the stability of the system. Meanwhile, the TICK can save a huge amount of memory spaces for miners.

D. Light client

In TICK, instead of downloading all the block headers, the light client only needs to download a preferred number of recent block headers. Therefore, the required storage space is constant and significantly less than conventional blockchains. Moreover, the proposed light client can validate transactions by requests proofs from the full node, and then makes it possible to perform fast payment on the light client.

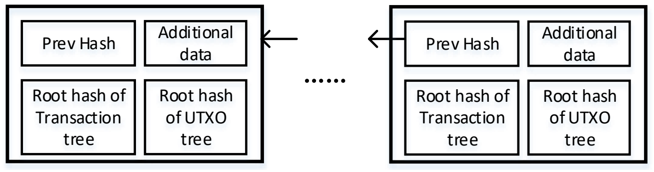

Fig. 7: Structure of Light client

The structure of the proposed light client is shown in Fig. 7. Instead of downloading all the block headers as the conventional light client, similarly with the miner proposed in section IV-C, the proposed light client only needs to download a preferred small number of block headers from a full node. In order to track the main chain, similarly with the proposed miner, the light client also needs to download at least one block headers from the main chain. Compared with the original light client, it has two advantages: 1. its storage requirement is constant and much less than the conventional light client; 2. it can validate transactions.

In TICK, to validate a transaction, the light client sends the objective output to the full node and gets the proof of PoP or PoA, and then checks the correctness of the responded proof. In more detail, as presented in Figure 8:

- Firstly, the light client gets the hash value β of the latest block header, and groups it with the hash value of the objective output that he wants to verify as a message *m*. Then the light client sends *m* to the full nodes.
- When the full node receives the message, it first locates the objective block according to the received β, and constructs the UTXO tree T_{UTXO} of the block. Since the UTXO tree is an ordered data structure, the full node uses a search algorithm to check whether the output is in the UTXO tree. If the output is in the UTXO tree, the full node generates the proof of presence PoP(T_{UTXO}, output) and replies the proof to the light client. If not, the full node generates the proof of absence PoA(T_{UTXO}, output) and replies the proof to the light client.
- When the light client receiving the proof, it first computes the root hash of the UTXO tree according to the received proof. If the computed root hash equals to the root hash stored in the block header. The light client considers that the receiver proof is correct, otherwise, the received proof is considered as wrong. In case of the received proof is right, the output is a UTXO if the received proof is PoP(T_{UTXO}, output), otherwise, it is not a UTXO.

After receiving all the proofs, for a transaction, if all the referred output are proved to be UTXOs, we compute the sum of values of them. If the sum of values of all the referred outputs is no larger than the sum of values of outputs, the transaction is valid.

During the transaction verification, the light client needs to communicate with the full node, therefore consumes bandwidths. When the light client sending message to the full node, the consumed bandwidth is the size of *m* which is $2*32$ bytes as the size of β and the hash value of the objective output are both 32 bytes. When the light client receiving messages from the light client, the consumed bandwidth is the size of the PoA or PoP. Assuming the number of UTXOs in the UTXO tree is n, the size of PoP is $\log_2 n * 32 + (\log_2 n)/8 + 8$ bytes and the size of PoA is double that of PoP, where $\log_2 n * 32$ is the size of critical hash values, $(\log_2 n)/8$ is size of the path and 8 is the size of the value. As a result, the size of PoA and PoP is relatively constant, and it increases 32 bytes every time the number of UTXOs doubled.

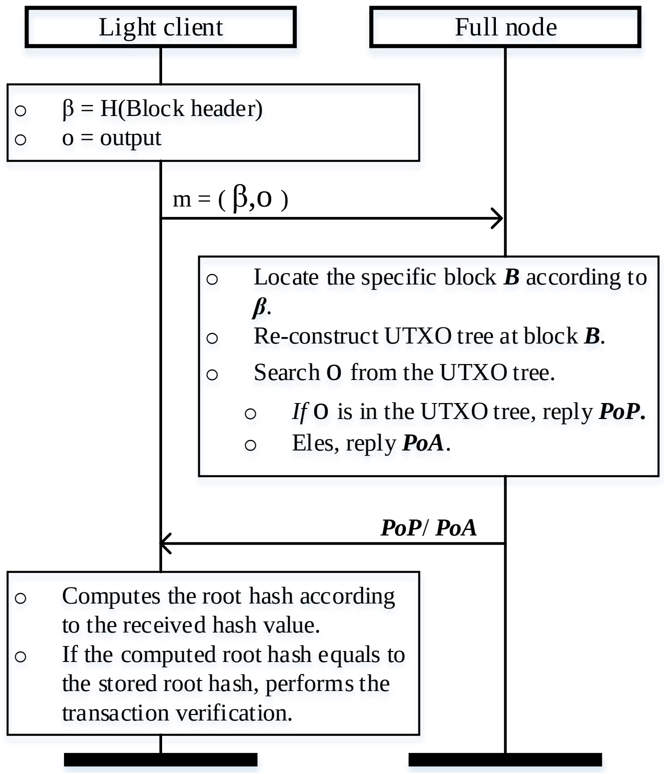
Light client
Full node
β = H(Block header)
o = output
m = (β,o)
Locate the specific block B according to β.
Re-construct UTXO tree at block B.
Search o from the UTXO tree.
If o is in the UTXO tree, reply PoP.
Eles, reply PoA.
PoP/ PoA
Computes the root hash according to the received hash value.
If the computed root hash equals to the stored root hash, performs the transaction verification.

Fig. 8: light client

V. Performance Evaluation

In this section, we implement TICK for the actual Bitcoin blockchain to evaluate its performance. Our evaluation is based on the Bitcoin data since its genesis block created in 2009 to date (29th March 2019), so when we mention now in this section we refer to March 29th, 2019. The evaluation mainly focuses on the reduced storage requirement for the miner and the light client. In addition, we also show the overhead incurred by our method and evaluate whether the overhead is reasonable in terms of the gained improvement. The overhead includes the storage overhead to the full node for storing added data and the bandwidth overhead to the light client during transaction verification.

Miner In TICK, in contrast with Bitcoin which needs a miner downloading the full transaction history, a miner only needs to download a fixed number of blocks and a UTXO tree. Since in Bitcoin a block is permanently recorded on the blockchain if there are five blocks before it, when we implement TICK on Bitcoin we assume a miner downloads 6 blocks. The comparison between the size of the data that a miner needs to download in TICK and the conventional Bitcoin blockchain is shown in Fig. 10. According to the Figure, the size of data that a miner needs to download in Bitcoin system is around 200 GB so far, which is far above the storage capacity of common mobile devices. More importantly, the size of the data increases fast with the development of the Bitcoin blockchain. Particularly, the size increases 29 percent in the past year, and with this development speed the size will be doubled in 3 years.

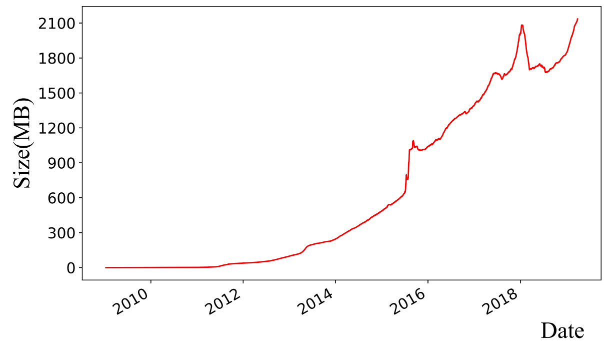

Fig. 9: UTXO size

In TICK, a miner needs to download all the UTXOs and 6 blocks. The change of block size is shown in Fig. 11. According to its change trend, we can see that the block size is around 1 MB in the past three years. Therefore, the total size of 6 blocks is around 6 MB which is very small and relatively constant. The change of the size of UTXOs is shown in Fig. 9. According to the Figure, by now the size of the UTXOs is around 2 GB. So totally, in our system a miner only needs to download 2 GB + 6 MB data which is significantly less than the data that a miner needs to download in Bitcoin. Therefore, when a new user joins the network as a miner, they only needs to download several GB of data rather than hundreds of GB data which can save lots of time and lots of storage space. More importantly, the size of UTXOs only increases 5 percentage in the past year which is significantly less than Bitcoin system's 29 percentage. As a result, the gained performance improvement of our method is becoming more and more over time. Consequently, TICK can attract more users to join the network as miners, and further makes the system more stable and securer.

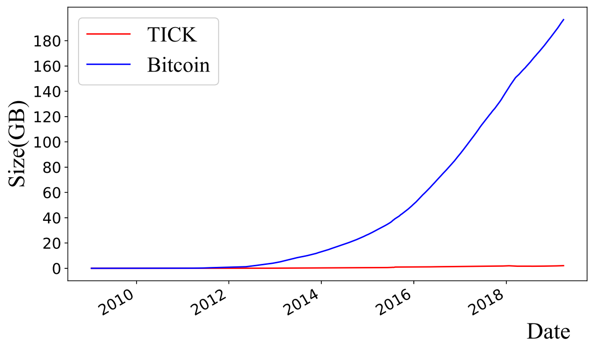

Fig. 10: Storage requirement of miner

Light client In Bitcoin, a light client should download all the block headers, and the number of block headers that a light client needs to download is 569339 by now. Furthermore, it needs to download one more block header every 10 minutes. In contrast, a light client in TICK only needs to download at least 6 block headers. Since the comparison between these two data sizes are trivial, we do not shown the comparison between them and the change trend of them. Apparently, TICK can save a large amount of memory space. More importantly, The size of data that a light client needs to download in our system is constant, which is important for storage-limited devices as it can avoid the additional storage consumption over time.

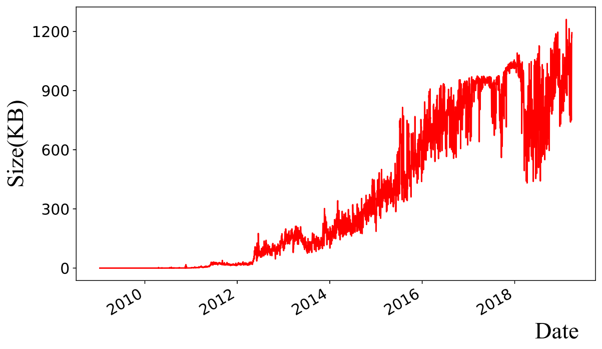

Fig. 11: Size of blocks

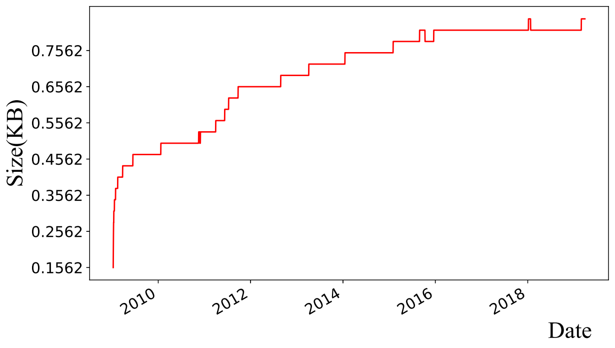

Fig. 12: Size of PoP

In Bitcoin, the light client can not validate transactions. But in TICK, since the full node uses a UTXO tree to maintain all the UTXOs and a full node can efficiently check whether an output is in the UTXO tree or not, the light client can validate transactions through request proofs from the full nodes. A light client first sends the output that he want to verified to a full node. If the output is in the UTXO tree, the full node replies the proof of PoP, otherwise the full node replies the proof of PoA. Since the size of PoA is twice of the size of PoP, we only show the size of PoP in Fig. 12. By now, the size of PoP is around 0.8 KB, and the size of proof of PoA is around 1.6 KB, which are both small. Moreover, the size of the proof is logarithm number of UTXOs, and it only increases one hash value (32 bytes) when the number of UTXO is doubled. According to the change trend of the UTXOs shown in Fig. 9, the size of UTXOs only increases 5 percentage in the past year, and with this speed the size of UTXOs will be doubled after 14 years. So the proof size is relatively constant in a long period.

Full node Our method brings significant performance improvement for light clients and miners. But for the full node, it may introduce the storage overhead. The full node needs to maintain the UTXO tree, and generates proof for light client. According to the size of UTXOs shown in Fig. 9, the size of UTXOs is 2 orders of magnitude less than the size of data of a full node (2GB against 200 GB). Therefore, the storage overhead caused by the UTXO tree is relatively slight. In addition, a full node needs to generate a proof for transaction verification. In the process, a full node needs to locate an output from the UTXO tree. In order to locate an element from a binary tree, it may need to search $\log_2 n$ times, where n denotes the number of elements in the tree. According to the number of UTXOs shown in Fig. 9, it at most needs to perform the search operation 27 times. The computational cost is relatively less compared with the computational cost of the puzzle solving. In addition, a full node is often deployed on a platform with high computational and storage capacity, so the computational cost and the storage cost is relatively slight for a full node.

VI. Conclusion

In conventional blockchains all the nodes maintain the full transaction history in order to keep the stable of the system. Therefore, the node should have a high storage capacity. In order to reduce the storage requirement, the light client which stores all the block headers is proposed. However, although it has less storage requirement, it can not validate transaction, and thus the light client can not perform fast payment. Therefore, users need to wait for six confirmations after its transaction is recorded. Actually, in the transaction history, lots of data are redundant for transaction verification, and only small portion of them, all the UTXOs, are useful when validating unconfirmed transactions. Therefore, we propose TICK which adds a new data structure, UTXO tree, to maintain all the UTXOs. With the UTXO tree, user can efficiently check whether an output is a UTXO or not and further validate transactions. In TICK, instead of maintaining all the block headers, a light client only needs to maintain a constant number of block headers, and it can validate transactions through sending request to a full node. Therefore, the data that a light client needs to maintain is constant and less.

As a side effect, miners in TICK do not need to download all the transaction history. It can start to mine while only downloads a preferred small number of blocks and the UTXO tree. The size of the download data is significantly less than the data of the whole transaction history. Therefore, TICK can help miners to save time and memory space, and further can attract more users joining the system as miners. Thus makes the system more stable and securer.

References

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

[2] "The internet of trusted things - kaleido insights," http://www.kaleidoinsights.com/reports/internet-of-trusted-things-blockchain/, (Accessed on 03/25/2018).

[3] A. Tapscott and D. Tapscott, "How blockchain is changing finance," *Harvard Business Review*, vol. 1, 2017.

[4] M. Mettler, "Blockchain technology in healthcare: The revolution starts here," in *e-Health Networking, Applications and Services (Healthcom), 2016 IEEE 18th International Conference on*. IEEE, 2016, pp. 1–3.

[5] R. Dennis and G. Owen, "Rep on the block: A next generation reputation system based on the blockchain," in *Internet Technology and Secured Transactions (ICITST), 2015 10th International Conference for*. IEEE, 2015, pp. 131–138.

[6] C. Chen, "The mathematically secure way to accept zero confirmation transactions," *Cryptocoin news*, vol. 13, 2014.

[7] J. Yu, V. Cheval, and M. Ryan, "DTKI: A new formalized PKI with verifiable trusted parties," *Comput. J.*, vol. 59, no. 11, pp. 1695–1713, 2016.

[8] J. Yu, M. Ryan, and C. Cremers, "DECIM: detecting endpoint compromise in messaging," *IEEE Trans. Information Forensics and Security*, vol. 13, no. 1, pp. 106–118, 2018.

[9] S. Delgado-Segura, C. Pérez-Sola, G. Navarro-Arribas, and J. Herrera-Joancomartí, "Analysis of the bitcoin utxo set," in *International Conference on Financial Cryptography and Data Security*. Springer, 2018, pp. 78–91.

[10] A. Palai, M. Vora, and A. Shah, "Empowering light nodes in blockchains with block summarization," in *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. IEEE, 2018, pp. 1–5.

[11] U. Nadiya, K. Mutijarsa, and C. Y. Rizqi, "Block summarization and compression in bitcoin blockchain," in *2018 International Symposium on Electronics and Smart Devices (ISESD)*. IEEE, 2018, pp. 1–4.

[12] A. Kiayias, N. Lamprou, and A.-P. Stouka, "Proofs of proofs of work with sublinear complexity," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 61–78.

[13] A. Kiayias, A. Miller, and D. Zindros, "Non-interactive proofs of proof-of-work." *IACR Cryptology ePrint Archive*, vol. 2017, no. 963, pp. 1–42, 2017.

[14] "Difficulty - blockchain." https://www.blockchain.com/en/charts/difficulty, (Accessed on 03/25/2019).

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

[16] S. Matetic, K. Wüst, M. Schneider, K. Kostiainen, G. Karame, and S. Capkun, "Bite: Bitcoin lightweight client privacy using trusted execution," IACR Cryptology ePrint Archive 2018, XXXX, Tech. Rep., 2018.

[17] K. Wüst, S. Matetic, M. Schneider, I. Miers, K. Kostiainen, and S. Capkun, "Zlite: Lightweight clients for shielded zcash transactions using trusted execution," in *International Conference on Financial Cryptography and Data Security. Springer*, 2019.

[18] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 281–310.

[19] L. Kiffer, R. Rajaraman *et al.*, "A better method to analyze blockchain consistency," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 729–744.

[20] R. Pass, L. Seeman, and A. Shelat, "Analysis of the blockchain protocol in asynchronous networks," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2017, pp. 643–673.

[21] "Ethereum blocks." https://etherscan.io/blocks/, (Accessed on 03/25/2019).

[22] "Blockchain charts: Bitcoin's blockchain size," https://blockchain.info/charts/blocks-size/, (Accessed on 03/25/2019).

[23] J. Herrera-Joancomart, "Another coin bites the dust: An analysis of dust in utxo based cryptocurrencies."