An Efficient Block Validation Mechanism for UTXO-based Blockchains

Xiaohai Dai*, Bin Xiao†, Jiang Xiao*, and Hai Jin*
*National Engineering Research Center for Big Data Technology and System
*Services Computing Technology and System Lab, Cluster and Grid Computing Lab
*School of Computer Science and Technology, Huazhong University of Science and Technology, China
†Department of Computing, The Hong Kong Polytechnic University, Hong Kong

Abstract*—It has been recognized that one of the bottlenecks in the UTXO-based blockchain systems is the slow block validation – the process of validating a newly-received block by a node before locally storing it and further broadcasting it. As a block contains multiple inputs, the block validation mainly involves checking the inputs against the status data, which is also known as the Unspent Transaction Outputs (UTXO) set. As time goes by, the UTXO set becomes more and more expansive, most of which can only be stored on disks. This considerably slows down the input checking and thus block validation, which can potentially compromise system security. To deal with the above problem, we disassemble the function of input checking into three parts: existence validation (EV), unspent validation (UV), and script validation (SV). Based on the disassembly, we propose EBV, an efficient block validation mechanism to speed up EV, UV, and SV individually. First, EBV changes the representation of status data, from UTXO set to a bit-vector set, which drastically reduces its size. The smaller status data can be entirely maintained in memory, thereby accelerating UV and also block validation. Second, EBV requires each transaction to carry the proof data, which enables EV and SV without accessing the disks. Furthermore, we also cope with two challenges in the design of EBV, namely transaction inflation and fake positions. To evaluate the EBV mechanism, we implement a prototype on top of Bitcoin, the most widely known UTXO-based blockchain, and conduct extensive experiments to compare EBV and Bitcoin. The experimental results demonstrate that EBV successfully reduces the memory requirement by 93.1% and the block validation time by up to 93.5%.

***Index Terms*—Blockchain, Bitcoin, UTXO, UTXO set, block validation**

I. Introduction

Due to its decentralization and anonymization merits, the blockchain system has received considerable attention from both academia and industry [1], [2]. Classified by the data models, the existing blockchain systems can be divided into two categories: Unspent Transaction Output (UTXO)-based and account-based [3]. Compared with the account-based model, the UTXO-based model provides a higher degree of privacy and concurrency, which is widely adopted by the cryptocurrency-like blockchain systems, with Bitcoin [4] as the representative. This paper mainly focuses on UTXO-based blockchains. Therefore, for the rest of the paper, unless stated otherwise, we refer to 'UTXO-based blockchain' as blockchain for brevity.

To ensure the security of the blockchain system, a block must be validated after it is newly received by a node. Only if a block passes the validation, will it be stored by the node locally and be further broadcast to other nodes. Since a block contains multiple inputs, the block validation process mainly involves checking the legitimacy of each input in it. Further, an input is considered legitimate if it can be found from a local database (status database), which stores all the Unspent Transaction Outputs (UTXO) and is also known as the UTXO set [5].

As time goes by, more and more outputs are accumulated in the UTXO set. To show the rapid expansion of the UTXO set clearly, we take the Bitcoin system as an example, depicting its change in the UTXO count and the size of the UTXO set by quarters, from the first quarter of 2015 (15-Q1) to the second quarter of 2021 (21-Q2). As shown in Fig. 1, the size of the UTXO set experiences a significant increase, which has exceeded 4.3 GB to date.

To prevent the blockchain system from occupying too many resources, the memory used by the UTXO set is usually restricted, especially in the resource-constrained nodes. For example, the memory usage of Btcd[1] (a Bitcoin client written in Golang[2]) is restricted to hundreds of MB, which is much less than the size of the UTXO set. As a result, the main parts of the UTXO set have to be stored in the slow disk, which slows down the input checking and thus block validation [6].

The low efficiency of the block validation can potentially compromise system security. First, only if a new block passes the block validation, will a node broadcast it to its neighbors. Therefore, the slow block validation can lead to a long propagation delay of blocks, which is a primary cause of the blockchain forks [7], [8]. From the perspective of a distributed system, the blockchain fork means inconsistent data stored in different nodes, which will endanger the security of a distributed system. Second, a newcomer must finish the Initial Block Download (IBD) to start a validator node [9]. During the process of IBD, each block is received from other nodes and validated locally to ensure its authenticity. If the block validation is slow, it will take a long time to finish the IBD process, which may discourage a user from starting a validator node. With a small number of validator nodes, the decentralization degree of a distributed system is low, thus

[1]https://github.com/btcsuite/btcd
[2]https://golang.org/

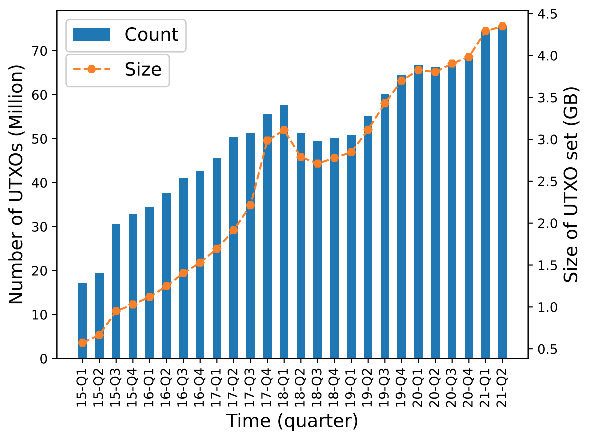


Fig. 1: Number of UTXOs and size of the UTXO set

compromising system security.

There are already some attempts trying to reduce the memory usage of the UTXO set or accelerate the block validation process. The first category works on the database layer, such as BZIP [10]. It compacts the keys and values stored in the database with shorter representations separately. However, memory pointers used by BZIP are temporary and will change once the node restarts, which makes this attempt impractical. Another category focuses on the protocol layer. One of the representatives is Edrax [11], which discards the UTXO set. However, it goes to an extreme to store almost nothing in a validator node, and totally relies on the transaction proposer to provide the proof. It requires each proposer to update its local proof in real time, which brings a heavy workload. Different from Edrax, our work stores a small amount of data in the validator nodes to facilitate the validation, which relieves the burden of transaction proposers.

To deal with the problem of slow block validation, we perform disassembly of the input checking process, which is the most critical part of the block validation. In general, the input checking is mainly used to check (1) if an input corresponds to an existent output (Existence Validation, EV), (2) if the output is unspent (Unspent Validation, UV), and (3) if the input can unlock the output (Script Validation, SV). According to our experimental analysis, the process of EV and UV takes up the largest parts of the block validation time, which inspires us to speed up the total block validation by accelerating EV and UV.

In this paper, we propose EBV, an efficient block validation mechanism for UTXO-based blockchains, which mainly changes the representation method of status data, from the UTXO set to a bit-vector set. Each vector in the bit-vector set corresponds to a block and each bit in a vector indicates if an output has been spent or not. Therefore, the bit-vector set can facilitate UV. Since the size of the bit-vector set is much smaller, which can be entirely maintained in memory, thus accelerating UV. Besides, EBV requires each transaction proposer to attach proof for each input in the transaction. Without relying on the status data, EV can be performed with the input proofs in memory, which is much faster. To sum up, the input checking (including EV and UV) can be conducted without accessing the slow disk, which improves the efficiency of block validation.

However, there are two challenges encountered in the above design. First, a new transaction has to embed a previous transaction as the input proof, which is used for SV. In this way, an old transaction will be included by a new one, and the new one will be further included by a newer one later, which leads to everlasting transaction nesting. Therefore, the transaction size is expected to be more and more inflated, which is named a *transaction inflation* problem. Towards addressing it, EBV separates the input data from the transaction and instead places its hash (input hash) in the transaction. A transaction with only input hashes is called a tidy transaction. Accordingly, the input proof in a new transaction contains the previous tidy transaction, without input proofs in the previous one, thus avoiding the *transaction inflation*.

Second, the input proof contains a *position* field to refer to the index of the to-be-spent output in the block. Since the *position* field is provided by the transaction proposer unilaterally, the malicious proposer may provide a *fake position value*. To tackle this challenge, EBV adds a *stake position* value for each transaction when packaging a new block, whose validity can be checked via the Merkle branch. Based on the *stake position* value, the *position* needed in the input proof can be acquired by calculation credibly.

To evaluate the efficient block validation mechanism, we implement a prototype of EBV on top of Bitcoin and compare it with Bitcoin. Since the input structure in Bitcoin is different from that in EBV, we also implement and run an intermediate node. The intermediate node synchronizes blocks from the Bitcoin mainnet, reconstructs the blocks as required in EBV, and then sends the reconstructed blocks to the EBV node. We are mainly interested in the performance of the EBV node. The experimental results demonstrate that EBV reduces the memory requirement largely, which is only 303.4 MB to date. When setting the memory limit as the same, EBV can reduce the block validation time and IBD time by 93.5% and 38.5% separately at most.

To sum up, we mainly make the following contributions:

- We make an experimental analysis of problems brought by the increasing size of UTXO set in UTXO-based blockchains, including the long block validation time and IBD time.
- We propose EBV, an efficient block validation mechanism for UTXO-based blockchains. Based on the local bit-vectors and proof attached in the inputs, EBV can check the legitimacy of blocks with little memory requirement.
- We implement a prototype on top of Bitcoin to evaluate EBV, whose experimental results demonstrate its low memory requirement and acceleration of block validation.

II. Preliminaries

In this section, we introduce the preliminaries related to our work, including the UTXO model adopted by mas-

TABLE I: Abbreviation Table

Short	Long	Short	Long
Ls	Locking script	*Us*	Unlocking script
MBr	Mekle Branch	EV	Existence Validation
UV	Unspent Validation	SV	Script Validation
Tx	Transaction	ELs	Enhanced Ls
IBD	Initial Block Download		
DBO	DataBase-related Operations		
UTXO	Unspent Transaction Output		

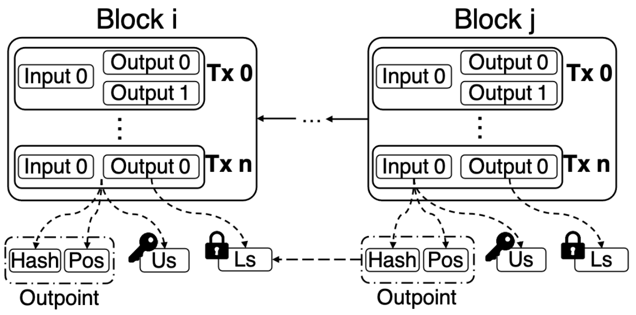


Fig. 2: UTXO model

sive blockchains and the UTXO set implemented by these blockchains. To aid reading, a table describing the abbreviations used in this paper is shown by Table I.

A. *UTXO model*

In terms of the data model, the blockchain systems fall into two categories, namely UTXO model [12] and account model [13]. Compared with the account model, the UTXO model has several advantages, such as parallel processing and a higher level of privacy [14], with Bitcoin as the best-known representative. In terms of parallel processing, the UTXO model enables multiple people to transfer coins in one transaction. The UTXO model also makes it harder to link transactions to a single user, thus protecting privacy better.

Fig. 2 illustrates the UTXO model, where plenty of transactions are packaged in a block. These transactions are logically organized as a Merkle tree, with each one as a leaf in the tree. A transaction is made up of various inputs and outputs [15]. Each output defines a Locking script (*Ls*), which locks the output and specifies the condition to unlock it. Each input attaches an Unlocking script (*Us*), which is used to unlock and spend a previous output. Particularly, *Ls* specifies the address eligible to spend the output, while *Us* proves the eligibility with a signature. An output without being unlocked is known as a UTXO [16]. To identify which output to spend, the input also contains a *hash* field and a *position* field. *Hash* field refers to the previous transaction creating the output, while *position* field indicates the position of the output in that transaction. The combination of *hash* and *index* is also named outpoint [17].

A block will be considered valid if and only if all the transactions in it are legitimate (block validation). Further,

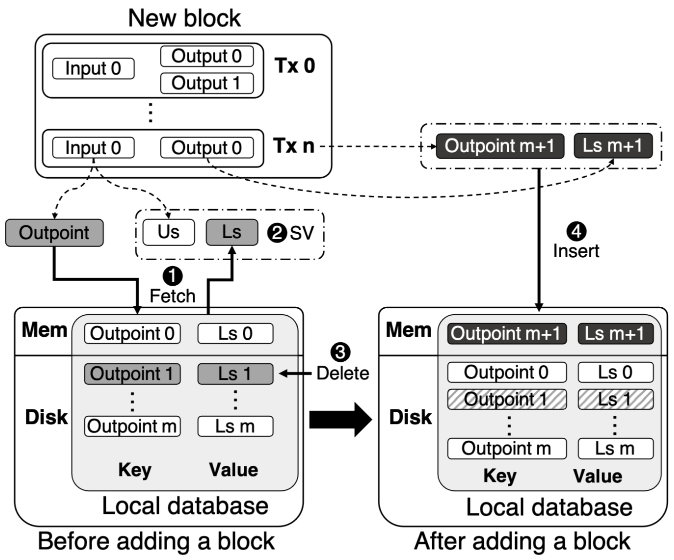


Fig. 3: UTXO set stored in the database

a transaction will be considered legitimate only if it passes various checks, especially the checking of inputs (input checking) [18]. Input checking includes three parts: 1) if the input corresponds to an existent output (Existence Validation, EV), 2) if the output has been spent (Unspent Validation, UV), and 3) if the execution result of *Us* plus *Ls* is true (Script Validation, SV) [19]. To be more specific, SV works through a stack-based scripting system, which typically checks if the public key contained in *Ls* matches the signature in *Us*, whose details can be found in [20].

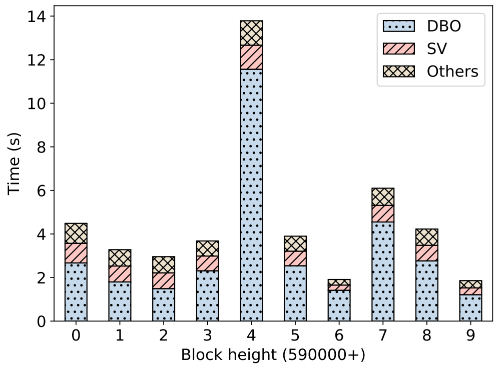
B. *UTXO set*

In the existing design of the UTXO-based system, the input checking involves accessing a local key-value database (e.g., LevelDB[3]), which maintains all the UTXOs and is also known as 'status database' [21].

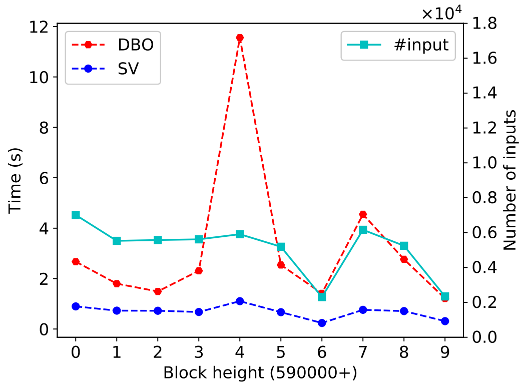
As shown in Fig. 3, the status database contains each UTXO as an entry, with the outpoint as the key and *Ls* as the value. Once a new block is received by a validator node, the outpoint of each input in the block is used to fetch the corresponding *Ls* from the database (i.e., ❶ Fetch). This operation exactly performs both EV and UV concurrently. If none is returned, the input and the block are considered invalid, which will be discarded later. Otherwise, the returned entry will be used for further validations, such as ❷ SV. If the block passes all the validations, the block will be considered valid. All the entries corresponding to the inputs in the block will be deleted from the UTXO set (i.e., ❸ Delete), and all the outputs will be inserted into the UTXO set as new entries (i.e., ❹ Insert). All of Fetch, Delete, and Insert are database-related operations (DBO), which are greatly influenced by the database efficiency.

However, as time goes by, the UTXO set has increased by a substantial margin. In retrospect of Fig. 1, the number of

[3]https://github.com/google/leveldb



(a) Different parts of validation time in Bitcoin



(b) Comparison of validation time and input count

Fig. 4: Time taken to validate a block in Bitcoin

UTXOs and the size of UTXO set in Bitcoin have increased by 4.4x and 7.6x separately. Therefore, it is impractical to store the UTXO set totally in the memory, especially for a resource-constrained node. As indicated in Fig. 3, the majority of the UTXO set is usually stored in the slow disk. To validate a new block, the memory will firstly be accessed to fetch the corresponding UTXOs. If not found, the disk will be further accessed. When most of UTXOs are stored in the slow disk, it usually takes a long time to perform the block validation.

III. Problem Analysis

In this section, we conduct several experiments to analyze the problems brought by the long block validation time. The experimental setup and hardware/software specifications are the same as that in Section VI, which will be introduced in Section VI-A later.

A. Long block validation time

After receiving a new block, a node will first perform the block validation on it. Fig. 4a takes Bitcoin as an example, depicting the validation time of ten blocks, whose block heights vary from 590000 to 590009. For each block, we divide the validation time into three parts: DBO (including Fetch, Delete, and Insert operations), SV, and others. From the figure, we can find that it takes several seconds to validate a new block. Particularly, for the block of height 590004, the block validation process even takes about 14 seconds. In addition, it is easy to conclude that DBO takes up most of the validation time. Also taking the block of height 590004 as an example, more than 83.8% of the total validation time is used for DBO.

What's more, we compare the number of inputs with DBO time and SV time in Fig. 4b. From the variation of three lines, it is easy to find that the variation of SV time is consistent with the number of inputs. However, the variation of DBO time is a little inconsistent, especially for the block of height 590004. Intuitively, DBO time is affected by two factors: the input number and the average database-access time. Since there is no large difference in the input number, the outlier of block 590004 should result from the inefficiency of the database. In other words, this outlier demonstrates that inefficiency of the database can possibly lead to a large DBO time and thus a long block validation time.

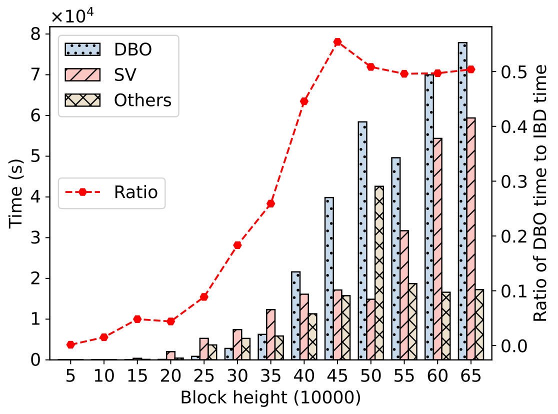


Fig. 5: Time taken to perform IBD in Bitcoin

B. Long IBD time

To study the relationship between block validation and IBD, we conduct some experiments on the IBD process of Bitcoin. From the genesis block to the block of height 650,000, the IBD process is divided into 13 periods, each of which contains 50,000 blocks. IBD time during each period is also divided into three parts: DBO, SV, and others, whose experimental results are shown in Fig. 5. In general, the time taken to perform DBO shows a rising trend, which is mainly caused by two aspects. On one hand, there is an increasing number of inputs in a block, which increases the frequency of DBO. On the other hand, due to the increasing size of the UTXO set, a single DBO may take much more time.

Besides, we also draw a line reflecting the ratio of DBO time to IBD time in Fig. 5. It is easy to find that the DBO time takes up over 50% of the total IBD time in the last five periods. In other words, the long DBO time leads to slow block validation and a long IBD process. What is interesting in the figure is the last but two period, namely from block 500,000 to block 550,000. Besides, both DBO time and IBD time experience a slight drop in this period. The reason for it is the transient reduction of the UTXO set, which is resulted from the rare consolidation actions [22].

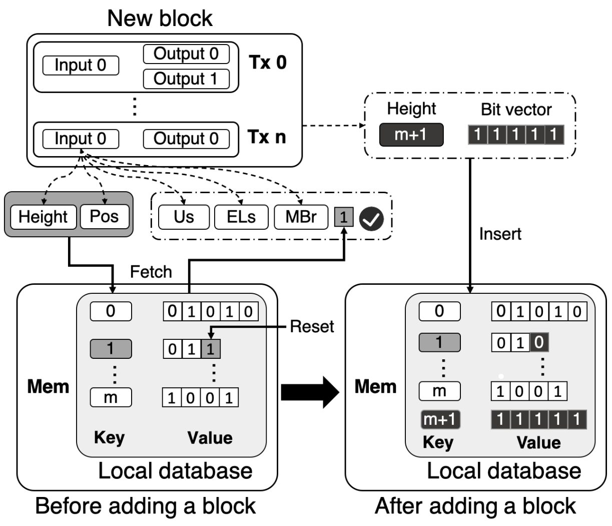


Fig. 6: System overview of EBV

IV. System Design

In this section, we elaborate on the design of EBV. In general, EBV changes the representation method of status data and the input structures. For the status data, EBV replaces the UTXO set with a bit-vector set. In terms of the input structure, EBV adds proof data for each input. Starting with a description of the threat model and an overview of EBV, we introduce the details about the transaction proposal, transaction validation, and block storage in EBV.

A. Threat model

We start by introducing the threat model. In general, our threat model is exactly the same as Bitcoin. Particularly, we assume that the computing power controlled by the adversaries is less than 50%. We also assume that the adversaries cannot break down the hypothesis of modern cryptography. In this regard, the common hash algorithms (e.g., SHA-256 [23]) used to build the Merkle tree and link the adjacent blocks cannot be cracked. Besides, we make the same assumption as almost all the mainstream blockchains that SHA-256 can resist collisions well. Further, we assume that the adversaries cannot tamper with data in remote nodes. In other words, if the data in a node can be modified by the adversaries, the node and its computing power are regarded as being controlled by the adversaries.

B. Overview

Fig. 6 depicts an overview of the EBV system. Comparing Fig. 6 and Fig. 3, the most important differences between EBV and the traditional blockchain (e.g., Bitcoin) include two parts: 1) local status database in a node; 2) input structure in the transaction.

Instead of storing all the UTXOs as in Bitcoin, EBV maintains a bit-vector set in the status database. The key in the database is a block height, while the value is a bit-vector indicating all the outputs in a block. In a bit-vector, each bit represents if the corresponding output has been spent or not.

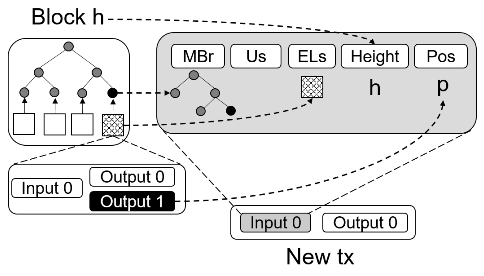


Fig. 7: Proposal of a new transaction

When a new block is appended to the chain, a new bit-vector is inserted into the database. If an output in this block is spent, the corresponding bit will be reset as 0. Since the size of a bit-vector is at most a few KB, the space requirement of bit-vectors for all the blocks is about hundreds of MB, which is much smaller than that of the UTXO set.

As stated in Section II-B, *Ls* is required to conduct SV. However, since no *Ls* is stored in the validator node's status database, *Ls* needs to be provided by the transaction proposer reliably. As shown in Fig. 6, apart from the *Us* as in Bitcoin, each input must be attached with an Enhanced Ls (*ELs*), a Merkle Branch (*MBr*) [24], a block *height*, and an output *position*. All of these will be detailed in the following sections.

C. Transaction proposal

This section describes how to propose a new transaction in EBV. Also, it deals with the challenge of transaction inflation encountered in this design.

1) Data structures of a transaction: As presented above, since less data is stored in a validator's status database, a transaction is requested to attach more data for validation. Toward this end, we introduce how to assemble a new transaction in EBV. A transaction mainly consists of two parts: inputs and outputs. Since there is no difference in the outputs part between EBV and Bitcoin, we focus on how to create the inputs part.

As shown in Fig. 7, an input in EBV includes five fields, namely *MBr*, *Us*, *ELs*, *height*, and *position*. *MBr* is a small part of the Merkle tree, which includes all the sibling nodes along the path from the tree root to the transaction containing the to-be-spent output. The *Us* field in EBV is the same as Bitcoin. Assume the to-be-spent output is contained in Block h, *height* field will be set as h. Besides, if the output is indexed as the p-th in Block h, the *position* field will be set as p. Compared with *Ls*, which only contains the scripts in an output, *ELs* is assigned the transaction containing the to-be-spent output. In other words, *ELs* is exactly the leaf node in the above *MBr*. It should be mentioned that although *Ls* is enough for SV, providing the entire transaction as *ELs* is necessary. The reason for it is that we make use of the *MBr* to do EV, whose hash calculation is done based on the entire transaction.

2) Transaction inflation challenge: However, taking the entire transaction as *ELs* also brings a challenge. Taking Fig. 8 as an example, transaction 'i' embeds transaction 'j' in its input, while the latter further embeds transaction 'k', and the

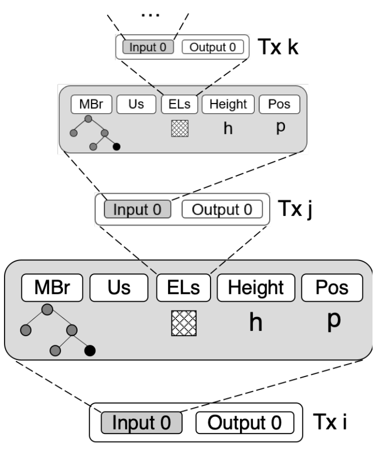
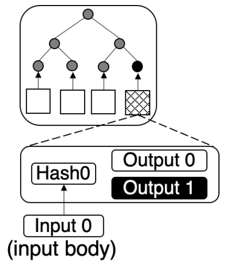
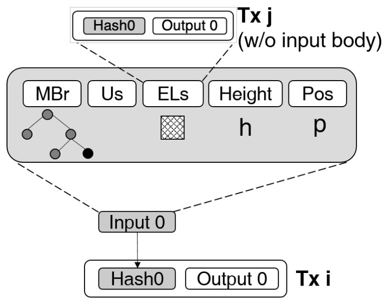


Fig. 8: Inflation problem of transactions



(a) Replace the input with its hash



(b) Input bodies of transaction 'j' are not needed

Fig. 9: Deal with the transaction inflation

like. The size of transaction 'i' is expected to be more and more inflated, which is referred to as a *transaction inflation* problem.

To address this challenge, EBV replaces each input in the transaction with the input's hash, as shown in Fig. 9a. Accordingly, only the hashes and outputs are involved in building the Merkle tree. However, the input bodies will also be transmitted and stored along with the transaction. This change of input structure will not invalidate the input checking. In particular, the outputs in *ELs* are enough for SV. Besides, EV is done by comparing two Merkle tree roots, one of which is stored in the validator's header, and the other is calculated based on the proof. Since the root in the header is created without input bodies when packaging a block, the proof also need not contain the input bodies. What's more, *ELs* has nothing to do with UV. Therefore, when taking transaction 'j' as *ELs* in transaction 'i', input bodies in transaction 'j'

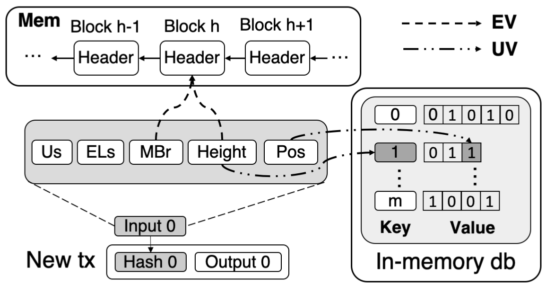


Fig. 10: Validation of a transaction

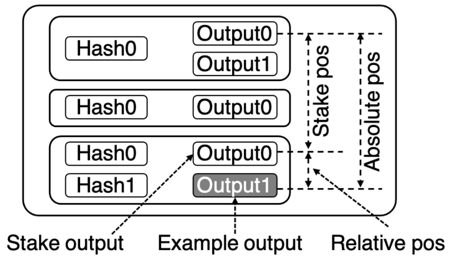


Fig. 11: Different positions of an output

are not necessarily required, as shown in Fig. 9b. Since the transactions (e.g., transaction 'k') embedded by the inputs of transaction 'j' are not needed in transaction 'i', the size of transaction 'i' is reduced largely, thus avoiding the *transaction inflation* problem.

D. Transaction validation

After receiving a transaction, a node has to validate the legitimacy of this transaction. Therefore, we describe how to perform the transaction validation in EBV and cope with the challenge of fake positions as well.

1) Process of transaction validation: As stated in Section II-A, the most important part of transaction validation includes EV, UV, and SV. Since the SV process in EBV works in the same way as the traditional ones, we focus on EV and UV in this section. As shown in Fig. 10, EV is conducted based on the *MBr* and *height* fields in the input, while UV relies on the *height* and *position* fields.

To be more specific, the validator will first find the specific header of height h in its local storage, and then figure out Merkle tree root based on *MBr* and *ELs*. Concretely speaking, hash values in *MBr* are calculated from the bottom up, with the top one as the *MBr* root. If the *MBr* root is consistent with that in the header, the output spent by this input will be considered as existent, thus passing EV. To conduct UV, the validator will first fetch the bit-vector from the database with the *height* as key. Then, the bit of index *position* in the bit-vector will be checked. If the bit is 1, the output will be considered as unspent, thus passing UV. Since both sizes of headers and bit-vectors are small, they can be totally stored in the memory, which speeds up the transaction and block validation largely.

2) Fake position challenge: Up to now, everything seems to work well, but in fact, we intentionally omit an important challenge: *fake position value*. Concretely speaking, since the *position* field is provided by the transaction proposer, what if the proposer provides a fake *position*?

Towards tackling this challenge, let's take a new look at the *position* value. As shown in Fig. 11, the *position* value we needed and mentioned before can be considered as an *absolute position* value, which is indexed from the first output in the whole block. Taking the last output in Fig. 11 as an example, its *absolute position* value is 4. From another point of view, an *absolute position* value can be regarded as the sum of two parts: a *stake position* value and a *relative position* value. Let the first output in the same transaction as the example output be a 'stake output'. *Stake position* represents the position value of the stake output, indexed from the first output in the whole block. In Fig. 11, *stake position* of the example output is 3. *Relative position* denotes the position value relative to the stake output, which is 1 for the example output. It should be noted that the input bodies are omitted from Fig. 11 to make the figures more readable.

Since all the outputs in a transaction are provided in the *ELs* field, *relative position* of an output can be easily acquired. EBV requires the miner to add a *stake position* field for each transaction when packaging a new block. With the *stake position* field, the *absolute position* of an output can be easily figured out. What's more, since *stake position* is contained in a transaction, its correctness can be guaranteed by *MBr*, which also prohibits the proposer from providing a fake position.

E. Block storage

If all the transactions in a block and further the block pass the validation, it will be stored locally by the validator node. In this part, we present the process of block storage, especially the update of the status database. Besides, we further optimize the size of sparse vectors in the bit-vector set, to further reduce the memory requirement.

1) Update of status database: The main process of block storage in EBV is similar to that in Bitcoin, except for updating the status database. In particular, the status database stores the UTXO set in Bitcoin, while maintaining the bit-vector set in EBV. Therefore, we detail the update of the bit-vector set in this section. As shown in Fig. 12, to store a new block, a new bit-vector indicating all the outputs in the block will be created and inserted into the database. Meanwhile, the bits corresponding to all the inputs will be reset as 0. If all the bits in a vector have been reset as 0, this vector can be deleted from the database, to reduce the memory requirement.

2) Vector optimization: As time goes by, more and more bits in a vector will be reset as 0. We refer to a vector with very few bits of 1 as a 'sparse vector'. A sparse vector can be represented by an index array of $\{i_0, i_1, ..., i_n\}$, where i_k represents a bit index of value 1, and n represents the count of 1-value bits. Taking Fig. 13a as an example, since there is only one 1-value bit in the 0-th bit-vector, it can be considered as a sparse vector. This sparse vector can be represented by an array of $\{3\}$, which indicates only the third bit has value 1. It is easy to find that only two bits (i.e., 11_2) are enough to

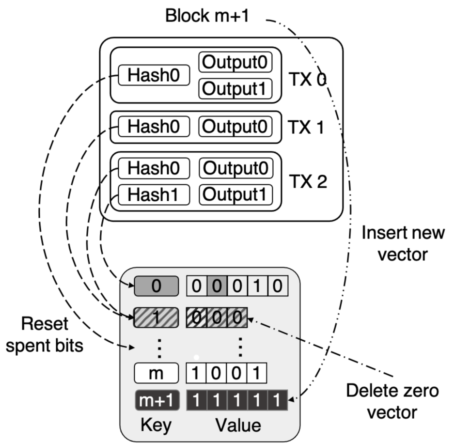
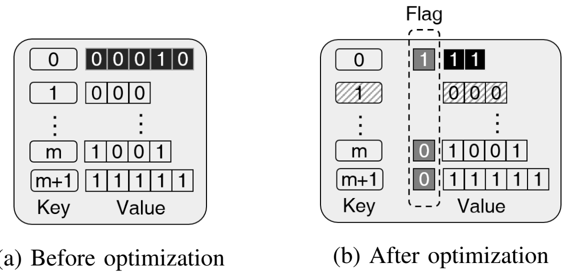


Fig. 12: Update of the bit-vector set



(a) Before optimization (b) After optimization

Fig. 13: Optimization of sparse vectors

represent the array of $\{3\}$, which is less than five bits in the original vector.

In the implementation, we add a flag bit in the front of each value to indicate the type of following bits, as shown in Fig. 13b. If the flag equals 0, the following bits represent a bit-vector. Otherwise, the following bits represent an index array. Since the number of outputs in a block is less than 65536, 16 bits are enough to represent an index in the array.

V. Security Analysis

On the whole, EBV takes the same security model as the mainstream blockchain systems (e.g., Bitcoin). It inherits almost all the working mechanisms from the traditional blockchain system, except that it provides an alternative structure to maintain the status data. In other words, mechanisms including locking/unlocking, PoW (Proof of Work) mining, and building of Merkle trees remain the same. The status data modified by EBV only changes the manner to check the validity of an input. Therefore, in this section, we mainly analyze the attacks trying to spend an invalid output, including spending a nonexistent output and spending an already spent output.

As for the first attack, the existence of an output can be verified through the *MBr*, by calculating the hash values from the bottom to the top. If the top hash value is the same as the specific root stored by a node, this output can be considered existent. Otherwise, the output is nonexistent, thus resisting the attack. In terms of the second attack, each node stores the bit-vectors locally to indicate if an output has been spent. Since an attacker is less likely to tamper with the bit-vectors in other nodes, the output can be prevented from being double-spent. To sum up, EBV can resist various system attacks, which is as secure as the traditional blockchain system.

VI. Evaluation

In this section, we conduct multiple experiments to evaluate our efficient block validation mechanism. Our experiments are based on Bitcoin, which is the ancestor of UTXO-based blockchains. Since the input structure in EBV is different from Bitcoin, the ledger data cannot be synchronized from Bitcoin to EBV directly. Therefore, we implement an intermediary node to reconstruct the input data, which is presented in Section VI-A. The evaluation metrics consist of four aspects: memory requirement, block validation time, IBD time, and propagation delay. In reality, we also considered the metrics of memory/disk bandwidth or the LevelDB performance indicators at the very start. Unfortunately, these bandwidth or performance indicators seem to be less helpful to demonstrate the conclusions. We repeat five times for each group of experiments to decrease the experimental errors. However, for the sake of space limitation and readability, we show only one of these results in most figures, if all the five-time experimental results verify the same arguments.

A. Experimental setup

All the evaluation of memory requirement, block validation time, and IBD time can be done in a small cluster, while the evaluation of propagation delay requires a large cluster. In this section, we mainly describe the common experimental setup for the former three metrics, leaving the specific experimental setup for the propagation delay in Section VI-E. The small cluster consists of three nodes, including an original Bitcoin node, an intermediary node, and an EBV node.

Each node contains an Intel(R) Core(TM) i7-6500U 2.50GHz CPU, 8 GB RAM, and 2 TB HDD, with Ubuntu 16.04 as the operating system. The Bitcoin node runs Btcd (v0.20.1-beta), which has already synchronized the entire chain data from the mainnet. The intermediary node is used to establish a new chain with inputs reconstructed, which will be elaborated on later. The EBV node is exactly the node applying the efficient block validation mechanism. Both the intermediary node and EBV node are implemented on top of Btcd.

The intermediary node firstly receives blocks from the Bitcoin node, where the former and latter play the roles of destination and source nodes respectively. However, instead of storing the blocks directly, the intermediary node reconstructs the blocks. To be more specific, a few fields are created for each input in the block, including *MBr*, *ELs*, *height*, and *position*. These new fields are combined with the corresponding input together as the new input. All the new inputs and other parts of the block are packaged as a new block, which will be stored by the intermediary node. As for the creation of *MBr*, the old block must be retrieved according to the input. Therefore, apart from storing the new block, the intermediary node also needs to maintain the relationship between inputs/outputs and blocks. Concretely speaking, we maintain a database to map from the input/output to the block height. With the block height, we can easily retrieve the block content from the chain.

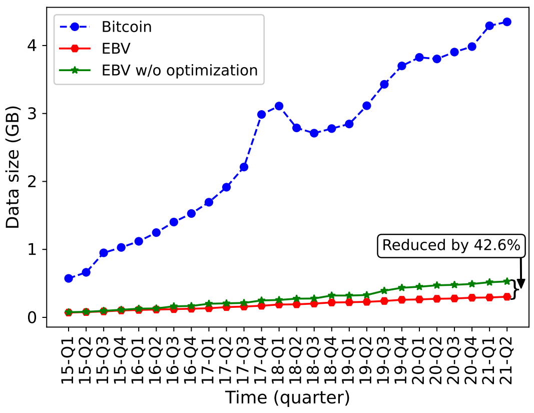


Fig. 14: Comparison of memory requirement

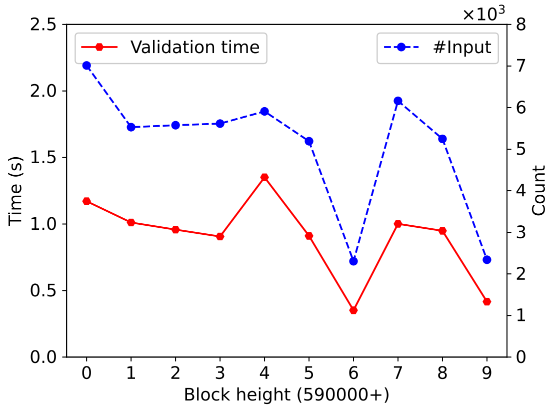


Fig. 15: Comparison of input count and validation time

After finishing the reconstruction of the new chain, the intermediary node plays the role of a source node, which synchronizes the new chain data to the EBV node. The synchronization process from the intermediary node to a destination node is exactly the one we make measurements. This process will be compared with that between two original Bitcoin nodes.

B. Memory requirement

In terms of the memory requirement, we only consider the part used to validate inputs. This part refers to the UTXO set for Bitcoin and bit-vector sets for EBV. In other words, the memory space used to store other data (e.g., block headers and unconfirmed transactions) is ignored in this section, since it is the same in both Bitcoin and EBV. To evaluate the effectiveness of vector optimization, we also measure the memory requirement of EBV without optimization.

Fig. 14 depicts the experimental results starting from 2015. It is easy to find that EBV reduces the memory requirement significantly. Particularly, in contrast to the 4.3 GB in Bitcoin, EBV occupies only 303.4 MB to date. Besides, by comparing EBV and EBV without optimization, we can conclude that the vector optimization approach brings large profits, which reduces the memory requirement by 42.6%. Furthermore, as time goes by, the vector optimization approach exerts a growing influence on memory reduction. The reason for it is that more and more outputs in a block will be spent, and the corresponding vector is more likely to be a sparse vector.

C. *Block validation time*

To evaluate the time taken to validate blocks, we select 10 blocks starting from block height 590000. In consideration of fairness, we set the value of memory limits as 500 MB for both Bitcoin and EBV. In fact, 500MB is much larger than the default value (100MB) hard-coded in Btcd and also the default value (450MB) set in Bitcoin Core[4] (another implementation in C++). Therefore, we argue that it is reasonable to set the memory limits as 500 MB in our experiments.

First, we compare the changing trends of the input count contained in each block and the corresponding block validation time, as shown in Fig. 15. From the figure, we can find that the variation of block validation time is roughly consistent with that of the input count. The reason for it is that all the data needed to validate a block has been stored in the memory. Without accessing the slow disk from time to time, the validation process can be totally done in the memory.

In addition, Fig. 16a compares the block validation time between Bitcoin and EBV. As expected, compared with Bitcoin, EBV greatly reduces the block validation time. To be more specific, as for the block of height 590004, EBV reduces the validation time by 93.5%. Furthermore, different parts of validation time in EBV are detailed in Fig. 16b, including EV, UV, SV, and others. It is easy to find that it takes little time to finish EV and UV, while most of the time is taken to do SV.

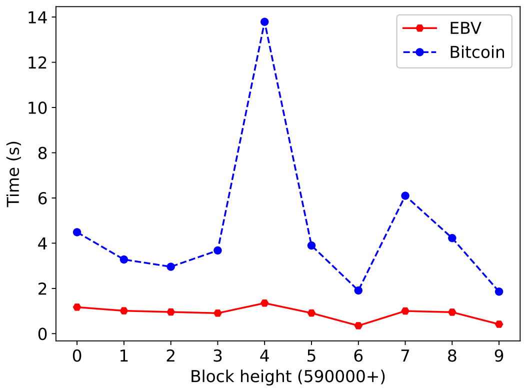
D. *IBD time*

We also evaluate the time taken to perform IBD. Similar to Section VI-C, the memory limit is set as 500 MB for both Bitcoin and EBV. The IBD time is recorded every 50,000 blocks, from the genesis block to the block of height 650,000.

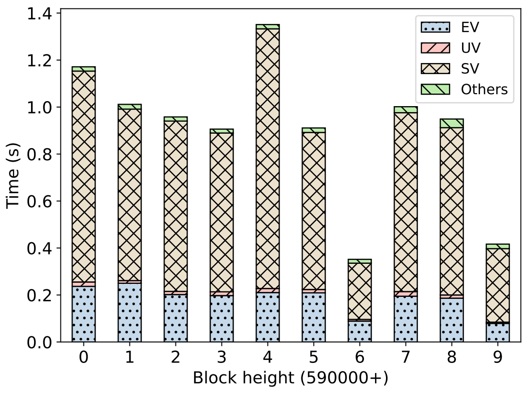
Fig. 17a makes a comparison of IBD time between Bitcoin and EBV, in the form of both boxplots and line plots. The boxplot describes the variations of five groups of experiments, while the line plot shows the average. By comparing the line plots, we can draw the conclusion that EBV is able to reduce the IBD time. In particular, by the block of 650,000, EBV reduces the IBD time by 38.5%. Besides, compared with Bitcoin, the IBD time of EBV increases more slowly. In other words, the larger is the number of blocks, the greater are the reduction effects brought by EBV. With regard to the boxplots, the variations are quite small both in EBV and Bitcoin, which demonstrate the stability of experimental results.

Detailed components of the IBD time are depicted in Fig. 17b. From the figure, we can find that both EV and UV

[4]https://bitcoincore.org



(a) Comparison of the validation time



(b) Different parts of validation time in EBV

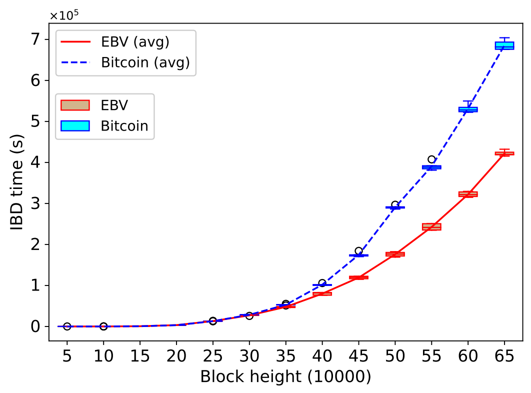
Fig. 16: Time taken to validate a block

take up a very small fraction of the total time, which also confirms the efficiency of the new block validation mechanism. Since it takes quite a large amount of time to do SV, interesting research works in the future may be the optimization of SV.

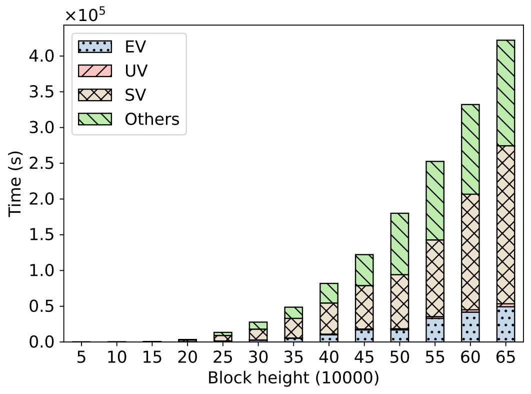
E. *Propagation delay*

Apart from the reduction of IBD time, the acceleration of block validation is also expected to reduce the propagation delay of blocks, thus lowering the risks of blockchain forks and enhancing system security. To demonstrate the improvement of block propagation brought by EBV, we further conduct the experiments to compare the propagation delays between Bitcoin and EBV. For each system, we deploy twenty nodes on AWS (Amazon Web Services) *t2.medium* instances dispersed in five regions and set the number of gossip neighbors in each node as two. We randomly pick a seed block and free it from a node. After that, the time to receive the seed block by each node is evaluated. The experiment is repeated five times to make the results more credible.

Experimental results are shown in Figure 18, which is consisted of boxplots and line plots. By comparing the line plots of the averages, we can conclude that EBV can exactly reduce the block propagation delays largely. Particularly, as for the moment when all the nodes receive the seed block, EBV can reduce the time by 66.4%. Besides, it is easy to find



(a) Comparison of IBD time



(b) Different parts of IBD time in EBV

Fig. 17: Time taken to perform IBD

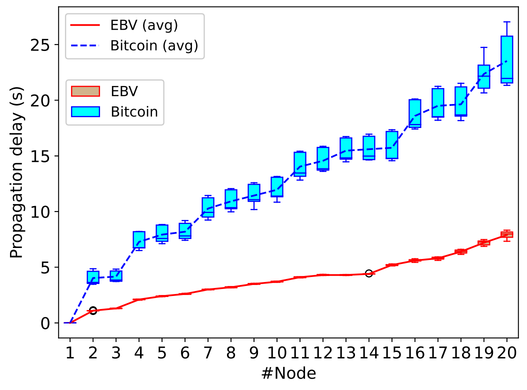


Fig. 18: Comparison of block propagation delays

that EBV has a lower variance than Bitcoin. The reason for it is that EBV maintains all the status data in the memory, while Bitcoin may maintain different parts of the status data in the memory at different times of experiments.

VII. Related Work

Although there are lots of works trying to improve blockchain performance from the perspective of consensus algorithms [25], [26], few of them give attention to the optimization of block validation, which can have important impacts on system security. According to different target layers, the small number of works to reduce memory requirement or accelerate block validation can be generally divided into two categories: database-layer and protocol-layer.

A. *Database-layer optimization*

Since the UTXO set is stored in a local database, a direct and natural idea is to optimize the operations on the database. The most representative one is BZIP [10], which adopts a data compression approach. By analyzing the keys and values stored in the UTXO set, BZIP identifies the redundancy problem of both key and value domains. To deal with it, BZIP designs two shorter representation methods for the key and value respectively, which require less memory. However, BZIP encounters two important problems in practice. First, it makes use of the memory pointer to represent a value domain. These memory pointers are volatile and will change completely if the node crashes and restarts. Second, the paper argues that few collisions are expected when taking a shorter representation method. In fact, however, most indexes in the shorter key representation by BZIP are distributed in a small interval, which makes the collisions very common.

B. *Protocol-layer optimization*

Instead of totally relying on the locally cached data to do validations, protocol-layer works make use of data from the transaction proposers. More precisely, in the system of this category, the transaction proposer attaches some proofs along with the transaction, which can be used for validation. These proofs are generated based on some novel structures, such as the sparse Merkle tree [27] and the accumulator [28]. Apparently, EBV also belongs to this category.

As a representative to adopt the sparse Merkle tree, Edrax organizes all the outputs in the tree, with each leaf in the tree as an unspent output or nil [11]. In fact, Edrax goes to an extreme to store almost nothing for the block validation in the validators. When proposing a new transaction, the proposer has to provide a branch of the sparse Merkle tree as proof. Each validator only needs to store the tree root, which is much smaller than the UTXO set. Although Edrax reduces the memory requirement in the validators, it brings an extremely heavy burden on the proposers. To be more specific, the tree root is updated in each block. To make the proof consistent with the tree root, each proposer has to update its local proof for each new block, which brings too large calculation overhead. What's worse, the size of the sparse Merkle tree is too large, whose tree height can reach 40. A branch in the large tree will be large too, leading to significant network overhead.

More schemes make use of the accumulators to create proofs, such as Utreexo [29], Boneh [30], and MiniChain [31] Each node in these schemes only stores the accumulator representation of the blockchain state, which is much smaller than the UTXO set. When proposing a new transaction, the proposer attaches proofs based on the accumulator. These proofs can be used by other nodes to validate the inputs in this transaction. However, these schemes also face some challenges. The size of proof in Utreexo has a positive relationship with the count of UTXOs. In other words, a proof may be larger and larger, as time goes by. As for Boneh, the dynamic addition and removal of elements in the accumulator may lead to its inefficiency. MiniChain requires the users to update the nonmembership witness in time. Otherwise, the updating of an old witness would be a time-consuming task, which brings a heavy burden on the users.

VIII. Conclusion

As time goes by, a large proportion of the UTXO set has to be stored in the slow disk, which reduces the efficiency of block validation. The inefficient block validation will further lead to a long delay of block propagation and a long time of initial blocks download, which may lower down the system security. To deal with these problems, we propose EBV, an efficient block validation mechanism, which reduces the memory requirement for block validation and speeds up the validation process. EBV can support various checkings of inputs, including EV, UV, and SV, which ensures system security. We conduct multiple experiments to evaluate EBV, whose experimental results demonstrate its reduction of memory usage and acceleration of block validation.

Acknowledgment

This work is supported by Hong Kong Research Grants Council General Research Fund PolyU 15216220 and 152124/19E, Key Research and Development Program of Hubei Province NO.2021BEA164, National Natural Science Foundation of China (Grant No.62072197), National Key Research and Development Program of China under Grant No.2021YFB2700700, the Technology Innovation Project of Hubei Province of China under grant No.2019AEA171, Key-Area Research and Development Program of Guangdong Province No.2020B0101090005.

References

[1] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey," *International Journal of Web and Grid Services*, vol. 14, no. 4, pp. 352–375, 2018.

[2] M. Swan, *Blockchain: Blueprint for a new economy*. " O'Reilly Media, Inc.", 2015.

[3] L. Brünjes and M. J. Gabbay, "Utxo-vs account-based smart contract blockchain programming paradigms," in *International Symposium on Leveraging Applications of Formal Methods*. Springer, 2020, pp. 73–88.

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

[5] D. McGinn, D. Birch, D. Akroyd, M. Molina-Solana, Y. Guo, and W. J. Knottenbelt, "Visualizing dynamic bitcoin transaction patterns," *Big data*, vol. 4, no. 2, pp. 109–119, 2016.

[6] S. Delgado-Segura, C. Pérez-Sola, G. Navarro-Arribas, and J. Herrera-Joancomartí, "Analysis of the bitcoin utxo set," in *Proceedings of the 22nd International Conference on Financial Cryptography and Data Security*. Springer, 2018, pp. 78–91.

[7] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *Proceedings of the 13th IEEE International Conference on Peer-to-Peer Computing*. IEEE, 2013, pp. 1–10.

[8] W. Hao, J. Zeng, X. Dai, J. Xiao, Q. Hua, H. Chen, K.-C. Li, and H. Jin, "Blockp2p: Enabling fast blockchain broadcast with scalable peer-to-peer network topology," in *Proceedings of the 14th International Conference on Green, Pervasive, and Cloud Computing*. Springer, 2019, pp. 223–237.

[9] "Initial block download," https://bitcoin.org/en/full-node#initial-block-downloadibd.

[10] S. Jiang, J. Li, S. Gong, J. Yan, G. Yan, Y. Sun, and X. Li, "Bzip: A compact data memory system for utxo-based blockchains," in *Proceedings of the 15th IEEE International Conference on Embedded Software and Systems (ICESS)*. IEEE, 2019, pp. 1–8.

[11] A. Chepurnoy, C. Papamanthou, and Y. Zhang, "Edrax: A cryptocurrency with stateless transaction validation." *IACR Cryptology ePrint Archive*, vol. 2018, p. 968, 2018.

[12] M. M. Chakravarty, J. Chapman, K. MacKenzie, O. Melkonian, M. P. Jones, and P. Wadler, "The extended utxo model," in *Proceedings of the 24th International Conference on Financial Cryptography and Data Security*. Springer, 2020, pp. 525–539.

[13] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[14] J. Zahnentferner, "Chimeric ledgers: Translating and unifying utxo-based and account-based cryptocurrencies." *IACR Cryptology ePrint Archive*, vol. 2018, p. 262, 2018.

[15] K. Okupski, "Bitcoin developer reference," in *Eindhoven*, 2014.

[16] N. Atzei, M. Bartoletti, S. Lande, and R. Zunino, "A formal model of bitcoin transactions," in *Proceedings of the 22nd International Conference on Financial Cryptography and Data Security*. Springer, 2018, pp. 541–560.

[17] R. O'Connor and M. Piekarska, "Enhancing bitcoin transactions with covenants," in *Proceedings of the International Conference on Financial Cryptography and Data Security*. Springer, 2017, pp. 191–198.

[18] V. Vallois and F. A. Guenane, "Bitcoin transaction: From the creation to validation, a protocol overview," in *Proceedings of the 1st Cyber Security in Networking Conference (CSNet)*. IEEE, 2017, pp. 1–7.

[19] A. M. Antonopoulos, *Mastering Bitcoin: unlocking digital cryptocurrencies*. " O'Reilly Media, Inc.", 2014.

[20] H. Brakmić, "Bitcoin script," in *Bitcoin and Lightning Network on Raspberry Pi*. Springer, 2019, pp. 201–224.

[21] "Data storage in bitcoin," https://en.bitcoin.it/wiki/Bitcoin_Core_0.11_(ch_2):_Data_Storage.

[22] "Utxo consolidation report," https://bitcoinops.org/en/xapo-utxo-consolidation.

[23] H. Gilbert and H. Handschuh, "Security analysis of sha-256 and sisters," in *International workshop on selected areas in cryptography*. Springer, 2003, pp. 175–193.

[24] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Proceedings of the Conference on the Theory and Application of Cryptographic Techniques*. Springer, 1987, pp. 369–378.

[25] M. Zhang, J. Li, Z. Chen, H. Chen, and X. Deng, "Cycledger: A scalable and secure parallel protocol for distributed ledger via sharding," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2020, pp. 358–367.

[26] L. Lao, X. Dai, B. Xiao, and S. Guo, "G-pbft: a location-based and scalable consensus protocol for iot-blockchain applications," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2020, pp. 664–673.

[27] R. Dahlberg, T. Pulls, and R. Peeters, "Efficient sparse merkle trees," in *Proceedings of Nordic Conference on Secure IT Systems*. Springer, 2016, pp. 199–215.

[28] J. Benaloh and M. De Mare, "One-way accumulators: A decentralized alternative to digital signatures," in *Workshop on the Theory and Application of Cryptographic Techniques*. Springer, 1993, pp. 274–285.

[29] T. Dryja, "Utreexo: A dynamic hash-based accumulator optimized for the bitcoin utxo set."

[30] D. Boneh, B. Bünz, and B. Fisch, "Batching techniques for accumulators with applications to iops and stateless blockchains," in *Annual International Cryptology Conference*. Springer, 2019, pp. 561–586.

[31] H. Chen and Y. Wang, "Minichain: A lightweight protocol to combat the utxo growth in public blockchain," *Journal of Parallel and Distributed Computing*, vol. 143, pp. 67–76, 2020.