TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 08/21 pp1037–1052
DOI: 10.26599/TST.2023.9010014
Volume 29, Number 4, August 2024

SnapshotPrune: A Novel Bitcoin-Based Protocol Toward Efficient Pruning and Fast Node Bootstrapping

Pengfei Huang, Xiaojun Ren*, Teng Huang, Arthur Sandor Voundi Koe, Duncan S Wong, and Hai Jiang

Abstract: Node synchronization is essential for the stability of the Bitcoin network. Critics have raised doubts about the ability of a new node to quickly and efficiently synchronize with the Bitcoin network and alleviate the storage pressure from existing full nodes to stockpile new data. Basic pruning and other techniques have been explored to address these concerns but have been insufficient to reduce node synchronization delay and effectively suppress the growth of synchronized data. In this study, we propose SnapshotPrune, a novel pruning and synchronization protocol that achieves fast node bootstrapping in the Bitcoin blockchain. Real Bitcoin historical data are leveraged to measure the synchronization time and monitor the network traffic during node bootstrapping. The protocol requires data downloads that are 99.70% less than Bitcoin Core, 81% less than CoinPrune, and 60% less than SnapshotSave, thereby saving 97.23% of download time. Findings show that the proposed design enhances the storage efficiency and reduces the node synchronization delay compared with existing techniques. We hypothesize that the efficiency of this protocol increases with the block height.

Key words: blockchain; Unspent Transaction Output (UTXO) pruning; snapshot; fast bootstrapping; synchronization

1 Introduction

The primary motive behind the blockchain is to bypass a central bank with peer-to-peer transactions[1, 2]. The extensive research and improvements to such technology have benefited several fileds, including information sharing[3], copyright protection[4], Internet of Things[5–7], smart health[8], social networking[9, 10], file storage[11], and privacy preservation[12, 13]. Bitcoin[14] and Ethereum[15] are two leading blockchain platforms that support various nodes and transactions. The blockchain provides an open, transparent, and unchangeable bookkeeping model wherein everyone can participate. However, such a concept is based on a decentralized and distributed network. Consequently, the stability of the blockchain network is dependent on the majority of honest and reliable nodes in a global distribution. Node synchronization technology is essential for the integration of new nodes into the network. Only by synchronizing the on-chain data to the local, rebuilding the ledger, and maintaining the same state as the blockchain network can new nodes participate in daily activities, such as consensus and transaction verification. The development of blockchain technology has led to exponential growth in on-chain data, necessitating existing nodes in the

• Pengfei Huang, Xiaojun Ren, Teng Huang, Duncan S Wong, and Hai Jiang are with Institute of Artificial Intelligence and Blockchain, Guangzhou University, Guangzhou 510006, China. E-mail: sawaxyy@gmail.com; renxiaojun@gzhu.edu.cn; huangteng1220@buaa.edu.cn; duncanwong_gzhu@163.com; haijiang_gzhu@163.com.
• Arthur Sandor Voundi Koe is with Institute of Artificial Intelligence and Blockchain, Guangzhou University, Guangzhou 510006, China, and also with Pazhou Lab, Guangzhou 510330, China. E-mail: sandorvoundi@gmail.com.
* To whom correspondence should be addressed.
Manuscript received: 2023-01-09 ; revised: 2023-03-05; accepted: 2023-03-12

network to store a large amount of ledger data and new nodes to expend considerable computing and storage resources to download and verify data. This scenario has drastically increased the minimum performance requirements of network nodes. The overwhelmed old nodes withdrawing from the network cause difficulties for new nodes with inadequate performance to join, resulting in a small number of collectives controlling the nodes and a significant increase in centralization. This outcome has a substantial detrimental effect on the network stability and reliability, and is highly unfavorable for the expanding application scenarios. Furthermore, in the new global economy, the ability to complete complex tasks in the shortest amount of time has become a critical research topic. Critics have raised doubts about the ability of a new node to quickly and efficiently synchronize with the blockchain network.

Most studies on node synchronization focus on basic pruning techniques, with the relatively straightforward approach[16, 17] of clipping the original consensus data that needs synchronization. This technique can reduce the growth rate to a logarithmic level while reducing the data volume, but is limited to a specific consensus algorithm or system, and cannot be widely applied. Another popular method is to use application data as the database for synchronization[18–21], which can significantly reduce the required amount of data. However, as the block height increases, the application data are also likely to reach massive volumes, and do not inhibit the growth rate. The volume, and growth rate of application data cannot be ignored, given that the blockchain network has rapidly expanded in the number of addresses and transactions. For example, the recent block height in Bitcoin has maintained more than 8×10^8 Unspent Transaction Outputs (UTXOs), with a volume exceeding 6 GB. However, recent research has continued using synchronized complete application data without accounting for aspects, such as its volume and growth. Thus, this study aims to second-prune the application data and suppress its expansion in addition to its use as synchronized data.

In this study, we investigate a novel approach to fast bootstrapping in the Bitcoin network. The aim is to reduce the storage and delay that a new Bitcoin node requires to synchronize with existing full nodes. Profound reforms are carried out over previous work, such as CoinPrune[18], and faster synchronization is achieved during a node's bootstrapping to enhance the existing blockchain platforms. This study proposes a dynamic pruning technique for the set of UTXOs based on features extracted from the blockchain metadata and data, rather than a naive approach that solely relies on consensus data (blocks). The novel protocol reduces the storage requirement threshold on new network nodes. Then, the UTXOs that transcend the pruning are aggregated as snapshots. The security analysis of the remaining set of UTXOs suggests the validity and the ability to verify such snapshots.

Real Bitcoin historical data are leveraged to measure the synchronization time and monitor network traffic during node bootstrapping. The protocol requires data downloads that are 99.70% less than Bitcoin Core, 81% less than CoinPrune, and 60% less than SnapshotSave, thereby saving 97.23% of download time. Findings show that the design enhances the storage efficiency and reduces the node synchronization delay compared with existing techniques. We hypothesize that the efficiency of this protocol increases with the block height. The importance and originality of this study lie in exploration of Bitcoin node synchronization, which is now a well-established issue of blockchain point of entry.

The contributions of this study are as follows:

• A pruning strategy is proposed for UTXO-type application data, significantly reducing the amount of data downloaded without weakening security guarantees.

• The synchronization scheme is improved based on the existing snapshot mode to adapt to the UTXO pruning strategy and ensure the integrity of results.

• Experimental evidence shows that UTXO pruning based snapshot synchronization considerably reduces storage and bootstrapping costs while suppressing growth.

2 Background and Related Work

2.1 Background

(1) Bitcoin network structure

Bitcoin adopts a peer-to-peer (P2P) network architecture based on the Internet. As such, computers in the same network are equal to the other, and each node provides network services. Nodes are not particular, and each connects to the other in a flat topology in the network. As a result, the P2P network has are no servers, centralized services, and hierarchical structures. The nodes of the P2P network interact and operate cooperatively, each providing external services while also utilizing those provided by other nodes. The P2P network is reliable, decentralized, and available.

(2) Node type

Although the Bitcoin P2P network nodes are equal, each node may have a different division of labor depending on its functions. Each Bitcoin node is an available collection of routing, blockchain database, mining, and wallet services. The full node maintains a complete blockchain with all transaction information. In the early stage of Bitcoin's development, all nodes were full node. The current Bitcoin Core client is also a full node.

Full nodes preserve a complete and up-to-date replication of the blockchain containing all transaction information. These nodes can independently construct and verify the complete blockchain, from the genesis block to the latest block. At the same time, full nodes can independently verify any transaction information without resorting to other nodes or other sources of information. Through the Bitcoin network, the full node obtains and merges a new block update containing transaction information into the local replication of the blockchain after verification.

(3) Full nodes synchronization

Once launched, a new node must find and connect to at least one other in the network to be able to participate in the cooperative operations. A pre-existing Bitcoin node in the network may randomly link to a new node. Generally, new nodes connect to known ones using Port 8333 and the Transmission Control Protocol (TCP). While establishing a connection, the node sends a "version" message with complete authentication content to initiate the "handshake" communication. The initial step after connecting a full node to a peer is to reconstruct a complete blockchain. Consider a scenario in which an entirely new node only has the genesis block statically embedded in the client program. This new full node must download all of the data from the genesis to the most recent block to synchronize with the network and reconstruct the whole blockchain.

The synchronization of the blockchain starts with sending the "version" message, which the node can obtain from its peers to determine how many blocks each party has to compare with that held by its blockchain. The peers then exchange a "getblocks" message containing the hash of the top block of their local blockchain. If a peer identifies that the hash it receives does not belong to the top block but to an old one, then its local blockchain is better than other peer nodes with longer blockchains. This latter group has more blocks than other nodes and can identify which ones need to be supplemented. The first 500 blocks available for sharing are identified, and the hashes of these blocks are propagated using the "inv" message. Nodes that lack these blocks can request the complete information through the "getdata" message and use the hash value in the "inv" message to confirm whether it is the correct requested block, and determine the missing blocks.

2.2 Related work

How can we implement fast bootstrapping? This scalability challenge have been addressed mainly from three different perspectives. Based on the aforementioned background knowledge, the data synchronization has three primary phases for a new node that wishes to join the blockchain network as a full node: (1) The new node resumes communication with several established full nodes; (2) The node downloads block data from the established full node and verifies its validity; (3) All the transactions from the genesis to the latest block are determined, and the validity is confirmed. If any fault occurs, the new node re-initiates communication with other established nodes and attempts to download other convincing data. In this section, we survey the state-of-the-art measures that reduce storage requirements and improve bootstrapping from the abovementioned perspectives.

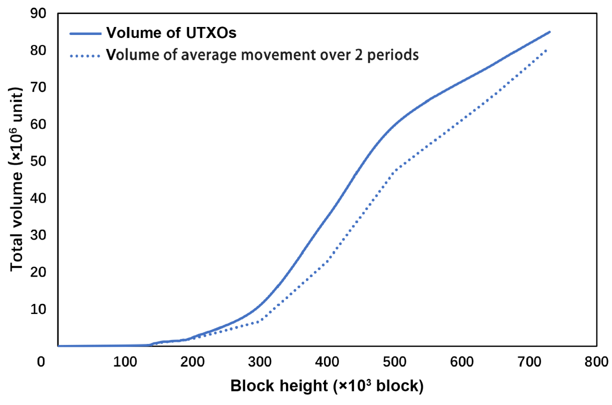
In most cases, reducing the amount of data to be downloaded also reduces information to be stored and maintained for newly joined nodes. Classical blockchain systems have identified this problem during the initial system design phase with methods to address the scalability challenge, such as Simple Payment Verification (SPV)[14, 22] and regular pruning operations. However, no long-term and satisfactory solution has been proposed to address the series of problems caused by the increasing on-chain data scale.The most attractive approach is sharding, or more specifically, network sharding, which, however, is difficult to achieve due to storage and computation[23–30]. This issue can be addressed by parallel and multiple chains. Inspired by data compression and shared storage in the distributed database, several researchers have attempted to reduce storage by encoding the on-chain data[31–33]. This approach aims to balance computation and storage. However, the actual performance savings are minimal due to the additional computation and time costs caused by encoding and decoding. In other words, this approach merely shifts the problem from one part to another.

Pruning useless or meaningless data[17, 21, 34–37] is likely the most straightforward and efficient method to reduce the amount of storage on the chain. At the same time, the object of pruning varies depending on the adopted ledger structure, such as the UTXO or account-balance model. Snapshots, meanwhile, are used to reduce the amount of data to be downloaded for synchronization[8, 18, 20]; these data have varying composition structures. Ensuring that the pruned data or snapshots can replace the original without compromising integrity and security, such as the ability to avoid double spending, is the most critical component of the two abovementioned approaches.

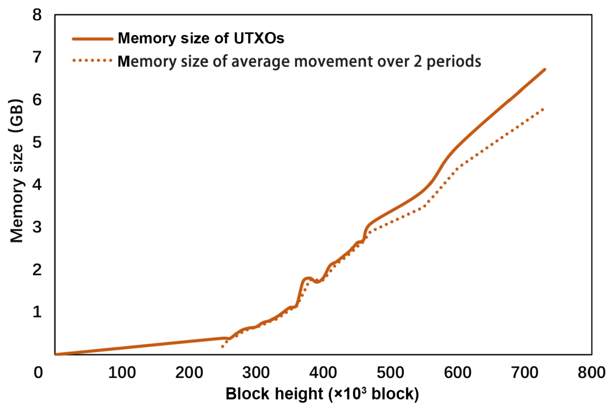
Overall performance improvements can also be achieved by optimizing a process within a bootstrapping strategy. For example, utilizing efficient resource allocations can improve network transmission efficiency and reduce latency[38–41], and more efficient cryptographic algorithms[42–44] or detection can reduce the time and computational overhead required for the verification. Additionally, reinforcement learning[45–47] can help to further optimize existing data synchronization strategies, and deep learning[48–50] can enhance the pruning of the current dataset.

3 Pruning Method

This study uses the UTXO model, as represented by Bitcoin. All nodes in the network maintain a local dataset called the chain state, which is dynamically updated in real time as the block height increases. Bitcoin is the earliest and most classic blockchain system and has been in existence for 13 years; thus its accumulated data is immense. This accumulation is mainly due to the explosive growth in the number of users and transactions, coupled with the stringent requirements for security and privacy at the application layer. The upper layer adopts a transaction and addresses obfuscation strategy, which generates considerable address and transaction data. In general, UTXO production and consumption are currently accelerating, and the total volume is gradually increasing. Figure 1 illustrates the approximate growth of the number and volume of UTXO sets corresponding to the height of the most recent block from the genesis block.



(a) Volume



(b) Memory

Fig. 1 Total volume and memory distribution of UTXO sets corresponding to the height of the most recent block from the genesis block.

Most cryptocurrency websites and other data analysis organizations carry out statistical analysis on sensitive and ubiquitous data structures in the blockchain system, such as block and transaction-related data. Within the scope of this study, a systematic and in-depth analysis of the meaning and regulation of UTXO itself is necessary. Additionally, further correlation analysis between UTXO and other data in the system is necessary. This section mainly expounds on the data analysis on the UTXO set and describes the pruning strategy based on the findings.

3.1 UTXO set

The Bitcoin-core client is downloaded to collect a reliable and comprehensive dataset and synchronize the most current blockchain data. Then, the chain-state database corresponding to a specific block height is built based on the local block file using the btcoin-cli-reindex command, and its file is parsed with the

UTXO-dump script to construct a standardized UTXO set. Without disrupting the regular website operation, we then retrieve the gathered spent block heights corresponding to each UTXO using the external data query method offered by blockchain.info. Additionally, btc.com is used to obtain more data for UTXO characteristics, such as Bitcoin price, block rate, and transaction fees.

We remove useless and non-integer features, such as tx_id, script, script_type, and address. Furthermore, we calculate the lifetime u_{life} of each UTXO, which is defined as the difference between the spent and generated heights. The final UTXO set includes 26 features and approximately 10^5 pieces of data.

3.2 UTXOs analysis

UTXO set analysis can determine the distribution characteristics of its lifetimes and influencing factors. Based on the qualities of the UTXO set, a preliminary analysis of the approximate distribution of UTXOs lifetimes is carried out using clustering algorithms (*k*-means, dbscan). Figures 2a and 2b show the results in the same block height range, which are roughly the same with a lifetime distribution bound of approximately 10^3 blocks for UTXOs. Only a small fraction of UTXOs remain for an extended period, and the vast majority of their lifetimes are shorter than 10^3 blocks. To obtain a comprehensive distribution, we re-collect approximately 5.4×10^6 newly generated UTXOs for the block height range of 700×10^3–701×10^3. Figure 2c shows that the clustering result confirms the presence of a dividing line at approximately 8×10^3 and 5×10^4 blocks. On this basis, we divide UTXOs into four groups according to the distribution of their lifetime, as shown in Table 1. UTXOs with lifetimes below 8×10^3 blocks are spent with a higher probability, particularly those within 10^3 blocks or less, than the other groups. By comparison, UTXOs with a longer distance are spent with a negligible probability, which becomes even smaller when the upper limit of 8×10^3 blocks increases.

In addition, we collect the UTXO set maintained at the height of 7×10^5 blocks and calculate the difference between the current and UTXO generation heights. Data statistics are then finally carried out according to the abovementioned four categories, as shown in Table 2. In the UTXO dataset maintained by the current block height, the number with a distance of over 8×10^3 blocks heights from the generated height accounts for

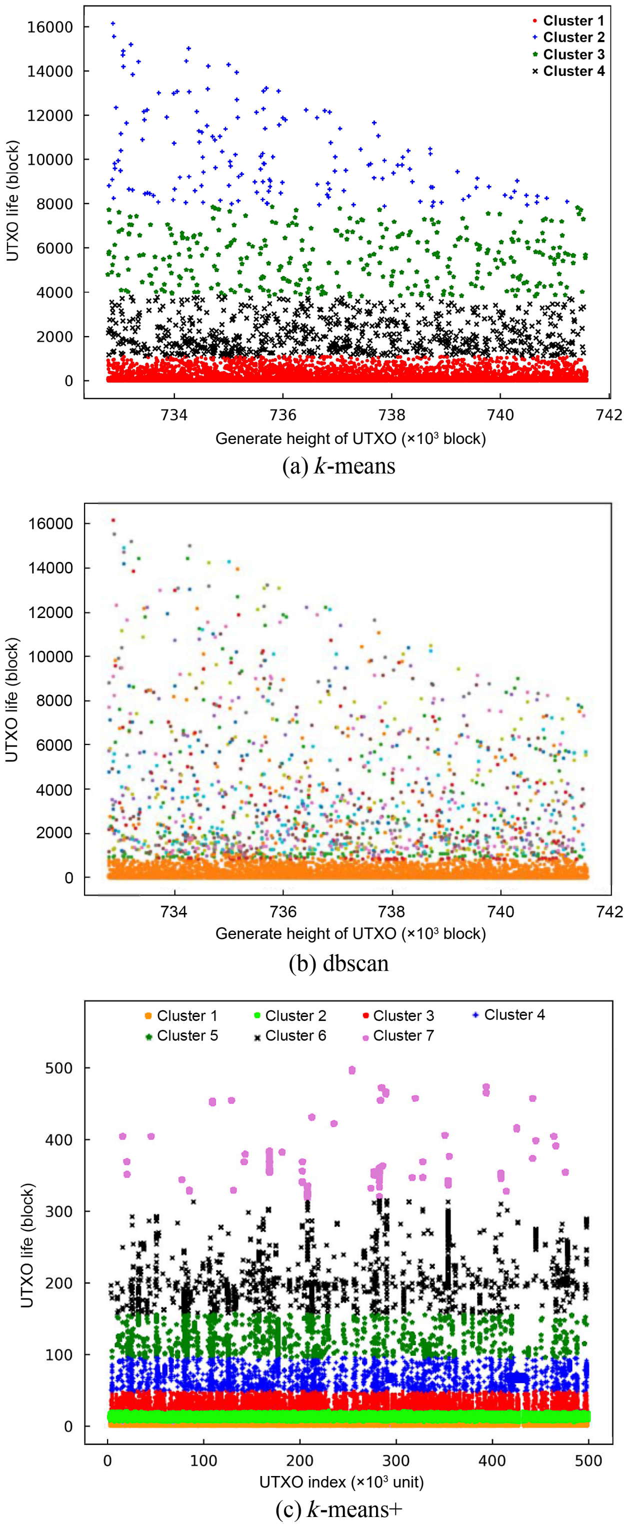


Fig. 2 Schematic representation of the clustering effect using different clustering algorithms (*k*-means and dbscan) for the UTXO lifetime set. In (b), the dbscan algorithm is used to generate the largest cluster (the orange part at the bottom) that differs significantly from other clusters.

Table 1 Specific distribution of UTXOs in one epoch of each classification.

Class	Block height range (block)	Proportion (%)	Average (block)	Standard deviation (block)	Median (block)
1	$0-1\times10^3$	77.32	105.76	176.85	26.00
2	$1\times10^3-8\times10^3$	13.10	3212.65	1843.72	2646.00
3	$8\times10^3-5\times10^4$	5.37	19 671.64	10 128.41	16 306.00
4	$\geqslant 5\times10^4$	4.21	89 229.88	50 852.19	69 065.00
Total	$\geqslant 0$	100.00	5309.59	21 052.20	73.00

Table 2 Statistical distribution of UTXOs at the current block height.

Class	Block height range (block)	Proportion (%)	Average (block)	Standard deviation (block)	Median (block)
1	$0-1\times10^3$	2.30	458.31	296.75	437.00
2	$1\times10^3-8\times10^3$	7.00	4297.63	2018.94	4319.00
3	$8\times10^3-5\times10^4$	19.82	25 149.52	11 711.84	23 543.00
4	$\geqslant 5\times10^4$	70.88	235 396.16	137 397.20	210 421.00
Total	$\geqslant 0$	100	172 145.00	152 239.47	139 948.00

more than 90%. However, the the nearest 8×10^3 blocks from the current height only generate a relatively small number of UTXOs, especially in the last 10^3 blocks that only account for 2.30%. Furthermore, we collect the UTXO set at the height of 7×10^5 blocks and calculate the difference between the current and the UTXO generation heights. We then carry out data statistics according to the abovementioned four categories, as shown in Table 2.

Combining the data distribution in Tables 1 and 2 shows that the update frequency of UTXOs with a generated height closer to the current one is much higher than those that are farther away. Moreover, in the set maintained by the current height, the former UTXOs have a much smaller proportion than the latter. Additionally, the number of UTXO sets that need to be maintained at the current height is as high as 8×10^8, most of which are to be spent later, resulting in a significant waste of storage space.

3.3 Pruning algorithm

The previous series of statistical analyses of the UTXO set reveal that a UTXO generation height that is close to the current block height leads to a high probability of being spent in the near future. In addition, the data to be maintained is a huge amount. Therefore, the most direct and effective method is to retain the most recently generated UTXO and prune those that have been held for a long time. Thus, this section mainly analyzes and solves the following problems: (1) how to determine the nearest and non-nearest boundaries, and whether the scope of the boundaries needs further refinement; (2) whether the boundary value is fixed or dynamic, and for the latter, what is the basis and strategy for the change; (3) as the block height increases, how to ensure the safe and reliable transmission and verification of critical information such as boundaries; and (4) what is the time point of pruning, block-by-block or periodic. A feasible pruning strategy is then designed.

The pruning is carried out periodically. The first block of the epoch inherits the boundary parameters of the previous epoch and dynamically maintains a spent UTXO life distribution in the data structure with the increase of block height. The law of UTXO spending selects the final division boundary for pruning and updates the boundary value. We embed the boundary and UTXO lifetime distribution information into blocks and construct an integer data structure with a size of 3×2 byte to store boundary values,

$$\mathcal{B}\{B_0,B_1,B_2\}\ (0<B_0<B_1<B_2<H_c).$$

where $\mathcal{B}$ denotes the set of boundary values, B_0, B_1 and B_2 denote three special boundary values from small to large, respectively, and H_c is the current block height. The initial settings are $B_0 = 10^3$, $B_1 = 8\times10^3$, and $B_2 = 5\times10^4$. Moreover, we define the four categories divided by three boundary lines as a distribution of UTXO lifetime $\mathcal{D}$, and the number of spent UTXO in each category is denoted by C_0, C_1, C_2, and C_3, respectively. A 4×3 byte integer data structure is used to store the lifetime distribution information of UTXO,

$$\begin{aligned}&\mathcal{D}\{C_0,C_1,C_2,C_3\},\\&C_0=\mathrm{sum}(\mathrm{life}[0,B_0]),\\&C_1=\mathrm{sum}(\mathrm{life}(B_0,B_1]),\\&C_2=\mathrm{sum}(\mathrm{life}(B_1,B_2]),\\&C_3=\mathrm{sum}(\mathrm{life}(B_2,H_c))\end{aligned}\tag{1}$$

where C_0 denotes the number of UTXOs with lifetime between 0 and B_0, C_1 denotes the number of UTXOs with lifetime between B_0 and B_1, and so forth for C_2 and C_3. sum ( ) is used to calculate the quantity of UTXOs within a special lifetime range given by life ( ).

With its unique nature, coinbase transactions do not require referencing the corresponding UTXO. Furthermore, there is no need for the unlock script for the transaction output. As such, the original unlocking script field can be filled with arbitrary data within a limited range. We embed the 18 byte boundary and lifetime distribution information into the coinbase transaction of each block. Furthermore, the lifetime distribution dynamic update Algorithm 1 shows how to maintain and update the above data structure and then transfer and verify the corresponding information between blocks to record the UTXO spending in the entire pruning epoch. The UTXO dynamic pruning Algorithm 2 demonstrates how to determine and update the final boundary value based on the critical information transmitted to the end block of the pruning epoch according to the UTXO set distribution regulation spent in the entire epoch.

Algorithm 1 Dynamic update of UTXO distribution

```
Require: $H_c$ : current block height;
    $Tx_c$ : previous block coinbase transaction;
    $Tx_n$ : set of current normal transactions to be blocked;
    $U_c$ : current UTXO set;
    $\tau$ : length of pruning epoch
Ensure: $B_c$ : current mining block
1: $j = (H_c + 1)/\tau$; // $j$ denotes serial number of epoch
2: $i = (H_c + 1) \bmod \tau$; // $i$ denotes serial number of block in $j$-th epoch
3: if $i \neq 0$ then
4:     $\mathcal{D}^j_{i-1}\{C^{i-1}_0, C^{i-1}_1, C^{i-1}_2, C^{i-1}_3\} \Leftarrow Tx_c$; // $\mathcal{D}^j_{i-1}$ denotes distirbution of spent UTXO lifetime of $(i-1)$-th block in $j$-th epoch
5: else
6:     $\mathcal{D}^j_{i-1}\{C^{i-1}_0, C^{i-1}_1, C^{i-1}_2, C^{i-1}_3\} \Leftarrow \{0, 0, 0, 0\}$;
7: end if
8: $\mathcal{B}_j\{B^j_0, B^j_1, B^j_2\} \Leftarrow Tx_c$; //$\mathcal{B}_j$ denotes boundaries of $j$-th epoch
9: for tx $\in Tx_n$ do
10:    inputs $\Leftarrow$ tx; //inputs denotes the set of tx inputs
11:    for input $\in$ inputs do
12:        Get transaction hash input$_{hash}$ used in input;
13:        Check hash in UTXO set $U_c$ and get output $u$;
14:        $u_{life} \Leftarrow H_c + 1 - u_{H_g}$; //$u_{life}$ denotes lifetime of spent UTXO $u$; $u_{H_g}$ denotes block height for generating UTXO $u$
15:        if $u_{life} \in [0, B^j_0]$ then
16:            $C^{i-1}_0 = C^{i-1}_0 + 1$;
17:        else if $u_{life} \in (B^j_0, B^j_1]$ then
18:            $C^{i-1}_1 = C^{i-1}_1 + 1$;
19:        else if $u_{life} \in (B^j_1, B^j_2]$ then
20:            $C^{i-1}_2 = C^{i-1}_2 + 1$;
21:        else
22:            $C^{i-1}_3 = C^{i-1}_3 + 1$;
23:        end if
24:    end for
25: end for
26: $\mathcal{D}^j_i\{C^i_0, C^i_1, C^i_2, C^i_3\} = \mathcal{D}^j_{i-1}\{C^{i-1}_0, C^{i-1}_1, C^{i-1}_2, C^{i-1}_3\}$;
27: if $i = \tau - 1$ then
28:    $\mathcal{B}_j \Leftarrow$ Execute Algorithm 2;
29: end if
30: Generate coin-based transaction $Tx'_c$ with $\mathcal{B}_j$ and $\mathcal{D}^j_i$;
31: Package $Tx'_c$ and $Tx_n$;
32: Create current mining block $B_n$
```

3.4 Pruning protocol

Beginning from the genesis block, we define τ as the pruning epoch, which means that UTXO set is to be pruned at τ-block intervals. The initial value of $\mathcal{B}$ is

$$\mathcal{B}_0\{B^0_0, B^0_1, B^0_2\} = \{1000, 8000, 50\,000\} \tag{2}$$

For the j-th epoch, the initial value of $\mathcal{B}$ is

$$\mathcal{B}_j\{B^j_0, B^j_1, B^j_2\} = \mathcal{B}_{j-1}\{B^{j-1}_0, B^{j-1}_1, B^{j-1}_2\} \tag{3}$$

While for the 0-th block in each epoch, the initial value of $\mathcal{D}$ is allways

$$\mathcal{D}^j_i\{C^0_0, C^0_1, C^0_2, C^0_3\} = \{0, 0, 0, 0\}.$$

Table 3 outlines the parameters involved in the pruning protocol. The basic flow is mainly divided into two stages, namely, parameters update and UTXOs pruning. Throughout the pruning period, each block height verifies and updates $\mathcal{D}^j_i$ according to the boundary parameter $\mathcal{B}_j$ of the current period based on Algorithm 1. At the end of the period block, according to the $\mathcal{D}^j_i$ of the entire epoch, the UTXO set is pruned, and the boundary parameter $\mathcal{B}_{j+1}$ for the next epoch is determined based on Algorithm 2. For the current block height H_c, the nodes in the network compete to mine the next block, which is the i-th block in the j-th epoch,

$$\begin{aligned} i &= (H_c + 1) \bmod \tau, \\ j &= (H_c + 1)/\tau \end{aligned} \tag{4}$$

Algorithm 2 UTXO set pruning algorithm

```
Require: $\mathcal{B}_j\ \{B_0^j, B_1^j, B_2^j\}$ : boundaries of $j$-th epoch;
    $\mathcal{D}_i^j\ \{C_0^i, C_1^i, C_2^i, C_3^i\}$ : distribution of $j$-th epoch;
    $U_c$ : current UTXO set;
    $\eta$ : pruning strength parameter;
    $\delta$ : update threshold of boundaries;
    $\varepsilon$ : maximum allowable error
Ensure: $U_p^j$ : pruned UTXO set in $j$-th epoch;
    $\mathcal{B}_{j+1}\ \{B_0^{j+1}, B_1^{j+1}, B_2^{j+1}\}$ : updated boundaries
1: Sum $= C_0^i + C_1^i + C_2^i + C_3^i$; // Sum denotes total number of UTXOs
2: $\mathcal{P}_j\ \{P_0^j, P_1^j, P_2^j, P_3^j\} = \mathcal{D}_i^j\ \{C_0^i, C_1^i, C_2^i, C_3^i\}$/Sum; //$\mathcal{P}_j$ denotes set of UTXO lifetime distribution ratios spent in the $j$-th epoch
3: if $P_0^j \geqslant \eta$
4:     $U_p^j \Leftarrow U_c\ [: B_0^j]$;
5:     if $P_0^j - \eta > \delta$
6:         $B_0^{j+1} \Leftarrow B_0^j \times (1 - \text{Random}\ [\delta, P_0^j - \eta])$; //Random [ ] denotes selecting a number from a range randomly
7:     else if $P_0^j - \eta \leqslant \delta$ and $P_0^j + P_1^j - \eta > \varepsilon$
8:         $B_1^{j+1} \Leftarrow B_1^j \times (1 - \text{Random}\ [\delta, P_0^j + P_1^j - \eta])$;
9:     else if
       $P_0^j - \eta \leqslant \delta$, $P_0^j + P_1^j - \eta \leqslant \varepsilon$, and $P_0^j + P_1^j + P_2^j - \eta < \varepsilon$
10:        $B_2^{j+1} \Leftarrow B_2^j \times (1 - \text{Random}\ [\delta, P_0^j + P_1^j + P_2^j - \eta])$;
11:    end if
12: else if $|P_0^j - \eta| > \delta$
13:    $B_0^{j+1} \Leftarrow B_0^j \times (1 + \text{Random}\ [\delta, |P_0^j - \eta|])$;
14:    if $P_0^j + P_1^j > \delta$
15:        $U_p^j \Leftarrow U_c\ [: B_1^j]$;
16:        if $|P_0^j - \eta| \leqslant \delta$ and $|P_0^j + P_1^j - \eta| > \varepsilon$
17:            $B_1^{j+1} \Leftarrow B_1^j \times (1 + \text{Random}\ [\delta, |P_0^j + P_1^j - \eta|])$;
18:        end if
19:    else if $P_0^j + P_1^j < \delta$ and $P_0^j + P_1^j + P_2^j \geqslant \delta$
20:        $U_p^j \Leftarrow U_c\ [: B_2^j]$;
21:        if $|P_0^j - \eta| \leqslant \delta$, $|P_0^j + P_1^j - \eta| \leqslant \varepsilon$, and $|P_0^j + P_1^j + P_2^j - \eta| > \varepsilon$
22:            $B_2^{j+1} \Leftarrow B_2^j \times (1 + \text{Random}\ [\delta, |P_0^j + P_1^j + P_2^j - \eta|])$;
23:        end if
24:    end if
25: else
26:    $U_p^j \Leftarrow U_c$;
27: end if
```

3.4.1 Parameters update

(1) Mine the acquired critical parameter information to be updated and stored in the coinbase data segment in the previous block coinbase transaction $\mathcal{B}_j\ \{B_0^j, B_1^j, B_2^j\}$ and $\mathcal{D}_{i-1}^j\ \{C_0^{i-1}, C_1^{i-1}, C_2^{i-1}, C_3^{i-1}\}$. Then the information correctness is verified.

(2) Traverse the transaction input to be packaged on the chain and acquire the referenced transaction hash and output sequence. Query the corresponding unspent transaction output , and calculate the life of the UTXO based on the currently maintained set,

$$u_{\text{life}} = H_c + 1 - u_{H_g} \tag{5}$$

(3) Determine the lifetime distribution interval of u according to the boundary parameter $\mathcal{B}_j$ and update $\mathcal{D}_i^j$ based on $\mathcal{D}_{i-1}^j$.

(4) Fill the boundary parameter $\mathcal{B}_j$ and the updated $\mathcal{D}_i^j$ into the coinbase data segment of the coinbase transaction of the mining block, and package the transaction to consensus.

3.4.2 UTXOs pruning

(1) At the end of the epoch, according to the boundary parameter $\mathcal{B}_j$, updated $\mathcal{D}_i^j$, and the pruning intensity, the final pruning boundary is determined according to the phases in Algorithm 2.

(2) Determine the next epoch boundary parameters $\mathcal{B}_{j+1}$ according to the precondition, and update function in Algorithm 2 based on $\mathcal{D}_i^j$,

$$B_i^{j+1} = B_i^j \times \left(1 \pm \text{Random}\left[\delta, \left|\sum_{n=0}^{i} P_n^j - \eta\right|\right]\right) \tag{6}$$

3.5 Pruning security analysis

The pruned UTXO set consists of snapshots to facilitate rapid node synchronization, which requires the assurance of fulfilling the same functions and roles as the full UTXO set. In other words, new nodes can operate as expected after successful synchronization with other full nodes. The UTXO set typically serves two primary purposes. First, forward validation is used to verify the completeness and legitimacy of the transaction flow in past blocks. Second, backward validation confirms the validity and traceability of the transaction input in future blocks.

The later spent UTXOs may no longer exist in the pruned set of UTXOs maintained by the current node, similar to how a Bitcoin node opens a block of memory locally to store a collection of isolated transactions. The parent transaction has yet to be packed onto the

Table 3 Parameters involved in the pruning protocol.

Parameter	Detail	Implication
$\mathcal{P}_j$	$\{P_0^j, P_1^j, P_2^j, P_3^j\}$	Collection of UTXO lifetime distribution ratios spent in the j-th epoch
$\mathcal{D}_i^j$	$\{C_0^i, C_1^i, C_2^i, C_3^i\}$	Set of UTXO lifetime distribution sets for the i-th block height spent in the j-th epoch
$\mathcal{B}_j$	$\{B_0^j, B_1^j, B_2^j\}$	Set of bounding parameters for the j-th epoch

blockchain for validation, and the legitimacy of child transactions cannot be confirmed. Consequently, a complementary set operation is required. If the current node receives a transaction with input that spends UTXOs that are not in the local set, then the node places this transaction into the isolated pool. Simultaneously, the node broadcasts a query request to other full nodes in the network. The responding nodes examine the locally maintained UTXOs set based on the transaction hash and the output sequence number list in the request and return the relevant packed UTXOs to the requesting node. By complementing the difference set operation, the synchronized node can verify subsequent transactions and act as a full node.

4 Bootstrapping Protocol

This chapter focuses on how to synchronize data when a new node joins the network using the pruned UTXO set discussed in the preceding chapter. The concepts of snapshot synchronization are adapted from SnapshotSave[20] and CoinPrune[18] with improvements to suit the present study objectives.

The general idea is to set a certain length of pruning and snapshot interval as one epoch, where miners execute the pruning algorithm outlined in Section 3 while generating a block. The final pruned UTXO set of is obtained and confirmed at the end of the current epoch. Based on the preceding pruned UTXO set, the miners create a snapshot at the first block of the subsequent epoch according to the data structure and method described in Section 5. Then, the final snapshot hash is generated and included in the proof of work.

4.1 Synchronization data structure

Data synchronization ensures that new peer nodes can autonomously reconstruct the system state at the current block height upon joining the blockchain network. On the one hand, new nodes can trace past blocks and transactions forward while having sufficient evidence to prove their legitimacy. On the other hand, these nodes can participate in the daily operations of the system with the same functions and authorizations as a full node, such as managing requests submitted by light nodes and linking new blocks locally. Therefore, functionality, integrity and validity must be ensured while requiring as little synchronized data content as possible. We follow CoinPrune's synchronization data structure, which comprises three main parts: header chain, snapshots, and tail blocks (shown in Fig. 3).

Header chain. Chain information refers to the block header information from the genesis block to the current snapshot epoch height. The block header contains information such as version number, previous block hash, Merkle tree root hash, timestamp, difficulty target, and nonce value. Combined with the UTXO set in each snapshot epoch, the above information can be reconstructed to the past chain state of the current snapshot epoch. Therefore, this information is necessary to put into the synchronization data.

Snapshot. Rather than consensus data (block set) in the blockchain network, we utilize application data (UTXO set) as the essential components of a snapshot. CoinPrune and SnapshotSave use periodic snapshots of UTXO set to reduce the number of data downloads necessary for synchronization. These snapshots are stored in multiple chunks with unit sizes of 1 MB and 3.5 MB, respectively. Furthermore, the corresponding number of UTXOs is approximately 1×10^4 and 5×10^4, respectively, constrained by the block and communication message sizes. Consequently, considering the order of magnitude of each snapshot ephemeris after pruning during the previous experimental analysis, we pack 5×10^4 UTXOs into a single chunk. The overall number of blocks is maintained at an order of magnitude of no more than one thousand bits, thus decreasing the frequency of communication. As in the case of SnapshotSave, we generate the corresponding Merkle tree based on the chunk of each UTXO set, and the root is then hashed with the block header information to obtain the final snapshot hash and incorporate it into the proof of work for verification.

Tail block. Tail blocks refer to all blocks' information from the end height of the most recent

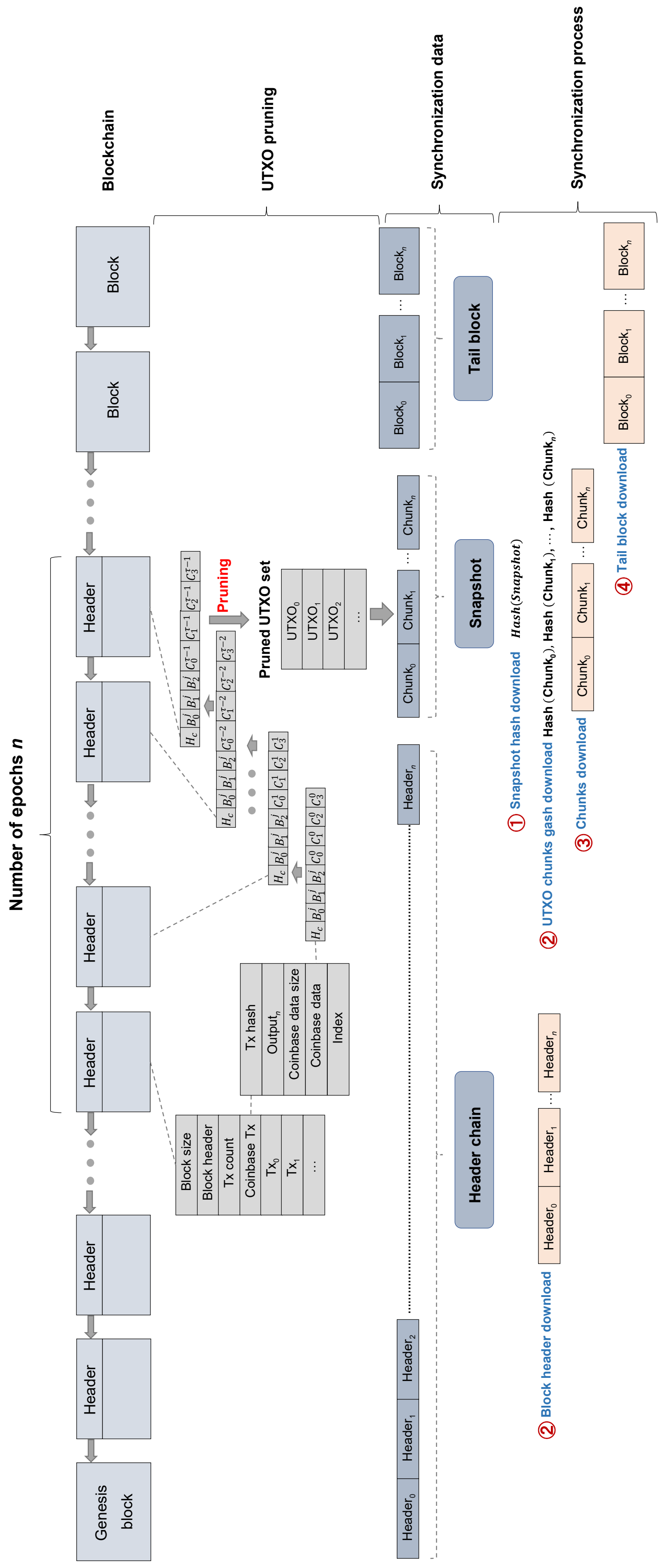


Fig. 3 Overview design of synchronization.

snapshot epoch to the current block height. Based on the UTXO set of the current snapshot epoch, the chain state from the previous snapshot epoch can be reconstructed by amalgamating the information of the closest complete blocks to the latest height.

4.2 Synchronization

The synchronization in this study is analogous to SnapshotSave, albeit with a few modifications. A new peer node completes the state synchronization in four steps: snapshot hash retrieval, UTXO chunk hash and block header download, chunks download, and tail block download (shown in Algorithm 3).

After establishing a connection with the old nodes, the new peer node requests the snapshot hash of the most recent epoch from its neighboring nodes and selects the most consistent reply. The correctness of the snapshot hash is then verified by its generation based on the hashes of the UTXO and block header chunks that correspond to the request. The chunks corresponding to the hash of UTXO chunk are requested, and their validity is checked. The header chain information is verified according to the original Bitcoin track, and the pruned UTXO set is obtained from the UTXO chunks to reconstruct the chain state prior to the latest snapshot. The tail blocks are then acquired to achieve a complete chain state reconstruction. The pruning strategy withdraws UTXOs that are unlikely to be spent in the future, and thus, old UTXOs may unavoidably be involved in subsequent transactions. Nodes must pack messages regarding these UTXOs and request downloads from neighboring nodes in the network. Similarly, if the reconstruction of chain state information is completed, this approach can also address the issue of unverifiable future transactions.

Algorithm 3 Pruning and synchronization

```
Require: $j$ : serial number of latest snapshot epoch;
    $\tau$ : length of pruning epoch;
    $\eta$ : pruning strength parameter;
    $\delta$ : update threshold;
    $\kappa$ : size of each UTXO chunk
1: for $i \in [j\tau,\ j\tau+\tau-1]$ do
2:     $\mathcal{D}_i^j \Leftarrow$ Execute Algorithm 1;
3:     if $i == j\tau+\tau-1$
4:         $U_{\mathrm{p}},\ \mathcal{B}_{j+1} \Leftarrow$ Execute Algorithm 2;
5:         $U_{\mathrm{p}}' \Leftarrow$ sort ($U_{\mathrm{p}}$); //$U_{\mathrm{p}}'$ denotes sorted $U_{\mathrm{p}}$; sort ( ) denotes function that sorts $U_{\mathrm{p}}$ by generation block height
6:         $C^j \Leftarrow \{U_{\mathrm{p}}'[0:\kappa),\ U_{\mathrm{p}}'[\kappa:2\kappa),\ \ldots,\ U_{\mathrm{p}}'[n\kappa:]\}$; //$C^j$ denotes set of UTXO chunks
7:         $\mathcal{H}_{\mathrm{c}}^j \Leftarrow \{\mathrm{Hash}\,(C_0^j),\ \mathrm{Hash}\,(C_1^j),\ \ldots,\ \mathrm{Hash}\,(C_n^j)\}$; //$\mathcal{H}_{\mathrm{c}}^j$ denotes set of chunk hash
8:         $\mathcal{S}^j \Leftarrow C^j\,\{C_0^j,\ C_1^j,\ \ldots,\ C_n^j\}$; //$\mathcal{S}^j$ denotes snapshot of $j$-th epoch
9:         $\mathcal{H}_{\mathrm{s}}^j \Leftarrow$ MerkleTreeRoot ($C^j$); //$\mathcal{H}_{\mathrm{s}}^j$ denotes snapshot hash of $j$-th epoch, MerkleTreeRoot denotes taking the value of root from merkletree of $C^j$
10:    end if
11: end for
12: For node $N$ at block height $h$, $h \in [(j+1)\tau,\ (j+2)\tau-1]$;
13: Blockheader $\mathrm{BH}^j \Leftarrow \{\mathrm{BH}_0,\ \mathrm{BH}_1,\ \ldots,\ \mathrm{BH}_{j\tau+\tau-1}\}$;
    // $\mathrm{BH}^j$ denotes set of block header BH from genesis to latest snapshot epoch
14: Tailblock $\mathcal{B}^j \Leftarrow \{B_{j\tau+\tau},\ B_{j\tau+\tau+1},\ \ldots,\ B_h\}$;
    //$\mathcal{B}^j$ denotes set of block $B$ from current snapshot epoch to latest block
15: Download latest snapshot hash $\mathcal{H}_{\mathrm{s}}^j$ and check validity;
16: Download chunks hash $\mathcal{H}_{\mathrm{c}}^j$ and blockheader $\mathrm{BH}^j$;
17: Request chunks $C^j$;
18: Request tailblocks $\mathcal{B}^j$.
```

5 Security Discusion

For the two main properties—namely, correctness and verifiability—that need to be satisfied by the pruning method proposed in CoinPrune, we carry out the corresponding security analysis based on the proposed pruning method and synchronization strategy. The security of the latter can be proven if the following two sets of conditions are met: (1) Correctness, which refers to the fact that each new node in the network must obtain an identical chain state to guarantee the integrity of accepted transactions; (2) Verifiability, which refers to the pruning strategy and must ensure that the new nodes can check the correctness of synchronization even in a complex network environment without compromising the security of the whole blockchain system.

Correctness. Based on the proposed pruning strategy, the full nodes in the network can be divided into two classes. One is the true full node N that holds the entire UTXO state set U, and the other is the slightly lighter full node N_{p} that contains the pruned UTXO state set U_{p}. UTXO updates are synchronized for N and N_{p} in each snapshot epoch, meaning U_{p} is always a subset of U. Additionally, whether based on U or U_{p} the result of continuing to perform pruning operation is equivalent. Thus, each newly joined node, has a shared core part, which we refer to as the pruned UTXOs set U_{pc}. However, for N_{p}, the U_{p} is not identical; that is, U_{p1} (the pruned UTXO set that one lighter full node generates) is not necessarily equivalent to U_{p2} (the pruned UTXO set another lighter full node generates). Furthermore, as transactions arrive, U_{p} can be further supplemented by U, and converge to U_{pm} (the largest collection of all U_{p}). In this process, the integrity of the accepted transactions is not compromised, and each node has the core chain state U_{pc}. Meanwhile, UTXOs outside the intersection with U_{pc} are not as essential for subsequent transactions.

Verifiability. To ensure the verifiability of the snapshots, we use the same strategy as in CoinPrune. The UTXO set can be verified by dividing the UTXO state set into data blocks and constructing the Merkle tree of the UTXO chunks. The snapshot hash is computed by combining the hashes of the Merkle tree root of the UTXO chunks and the block header. This snapshot hash is included in the Proof of Work (PoW) consensus of the coinbase transaction to receive cumulative confirmation from other nodes in the network, thus guaranteeing the verifiability of the snapshot. Nodes joining the network can verify the snapshot validity based on the reconfirmation of the most recent snapshot by the trailing block. Malicious nodes or miners cannot trick new nodes into accepting a changed or different snapshot, which is also challenging to construct.

6 Performance Evaluation

This section focuses on simulated experimental analysis that compares the proposed pruning strategy and enhanced snapshot approach with those of Bitcoin Core, CoinPrune, and SnapshotSave. First, we outline the environment setup for testing. Then, the storage savings from the different methods are presented. Finally, we empirically demonstrate that the number of downloads and synchronization time required for new nodes are significantly reduced.

6.1 Tested setup

We carry out a preliminary simulation to validate an implementation based on original Bitcoin data. The measurements are executed on a server (Intel(R) Xeon(R) Gold 6240R @2.40 GHz, 32 GB RAM, 60 TB hard drive) with synchronized data from personal computers (12th generation Intel(R) Core(TM) i5-12600KF @3.70 GHz, 32 GB RAM, 1 TB hard disk). The data are measured from the genesis block to the latest snapshot height, with 500 tail blocks. Bitcoin Core, CoinPrune, SnapshotSave, and the proposed method are monitored for data synchronization. The blockchain data determine the storage requirements, but the synchronization time and traffic may vary depending on the network state.

6.2 Storage savings

The amount of data to be downloaded and maintained by different node synchronization is obtained through simulations based on real Bitcoin historical data. Bitcoin Core necessitates downloading the complete blockchain, including consensus data such as all blocks and headers (block folder), and application data such as UTXO state (chain state folder). CoinPrune requires downloading a serialized snapshot (including the complete UTXOs data), a serialized header chain, and a series of tail blocks. SnapshotSave builds on CoinPrune and somewhat reduces the required amount of UTXO data to compose a snapshot. Then, the synchronization strategy further prunes UTXOs, which theoretically necessitates an even smaller amount of data to be synchronized. Experiments show that the proposed synchronization strategy requires 1.33 GB of data to be downloaded, as shown in Fig. 4a. In the overall trend and ignoring detail differences, this amount is approximately 99.70% less than that of Bitcoin Core (435.57 GB), 81% less than Coinrune (6.97 GB), and 60% less than SnapshotSave (3.33 GB). Moreover, as the blockchain grows, the savings may even increase at high block heights.

6.3 Evaluation of synchronization performance

Bitcoin Core necessitates downloading and verifying the block header and blocks, replaying the entire transaction (see Section 3.1). The other methods only require downloading and validating snapshots and tail blocks, except the block header. Depending on the amount of data to be downloaded for different synchronization strategies in the previous section, the corresponding synchronization times follow a similar trend. As illustrated in Fig. 4b, the average synchronization times consumed by the proposed synchronization strategies is approximately 10 minutes

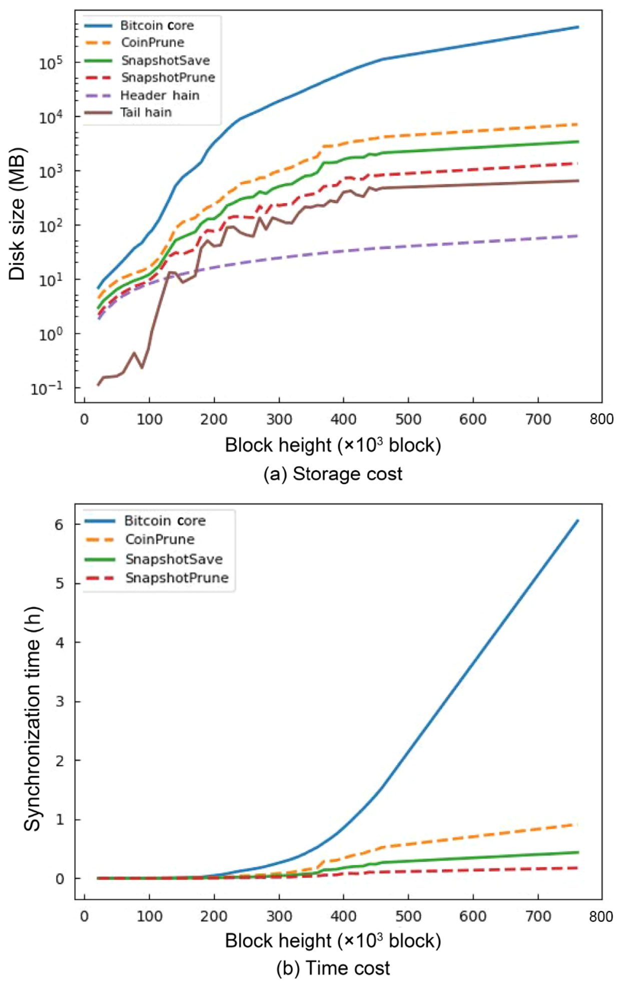


Fig. 4 Approximate distribution of storage capacity and synchronization time required by different synchronization strategies with the increase of block height.

(latest block height), whereas Bitcoin Core takes approximately six hours.

6.4 Evaluation of synchronization data accuracy

This study evaluated the accuracy of the pruned UTXO set using real historical data from Bitcoin. The accuracy rate was defined as the ratio of the inputs of future transactions that pruned UTXO sets can independently verify to the total transaction inputs, with the exception of UTXOs generated after the relevant block height at the end of the epoch. The pruning and snapshot epoch lengths are varied as 500, 1000, and 2000 block heights, and a specific block height is designated to mark the beginning of an epoch. We obtained the life distribution $S^j \Leftarrow C^j\,\{C_0^j, C_1^j, \ldots, C_n^j\}$ and corresponding proportion S^j of UTXO spent during each epoch and acquired the UTXO set j maintained by the corresponding block height at the end of the epoch. By adjusting the pruning strength parameter, the corresponding pruned UTXO set data $H_s^j \Leftarrow \text{MerkleTreeRoot}\,(C^j)$ is obtained. We then traced block transaction chains backward and counted the number of old transaction inputs that H_s^j can independently verify, computing the corresponding accuracy rate. The results in Table 4 demonstrate that the pruned UTXO set has considerable accuracy when the pruning strength is sufficiently high. Given the limitations of time and hardware performance, we choose a small number of future blocks for validation experiments. Still, the accuracy rate increases with the block height, eventually approaching an asymptote of value 1.

Table 4 Corresponding accuracies of the synchronization data with the effect of different parameter values: Length of each epoch τ, UTXO life distribution $\mathcal{D}$ and proportion $\mathcal{P}$, pruning boundaries $\mathcal{B}$, complete UTXO set $\mathcal{U}$, pruning strength η, and pruned UTXO set U_{p}.

τ (block)	$\mathcal{D}$ (UTXO)	$\mathcal{P}$	$\mathcal{B}$ (block)	$\mathcal{U}$ (UTXO)	η	U_{p} (UTXO)	Proportion (%)	Accuracy
500	$D_0 = 1\,817\,183$	$P_0 = 0.7673$	$B_0 = 1\times10^3$	52 082 148	0.95	24 186 818	46.44	0.956 822 97
	$D_1 = 398\,007$	$P_1 = 0.1681$	$B_1 = 8\times10^3$		0.90	7 324 361	14.06	0.867 155 33
	$D_2 = 117\,589$	$P_2 = 0.0497$	$B_2 = 5\times10^4$		0.80	7 324 361	14.06	0.867 155 33
	$D_3 = 35\,374$	$P_3 = 0.0149$			0.70	1 530 962	2.93	0.675 871 07
1000	$D_0 = 3\,858\,222$	$P_0 = 0.7485$	$B_0 = 1\times10^3$	51 953 149	0.95	23 439 410	45.15	0.912 630 7
	$D_1 = 845\,941$	$P_1 = 0.1641$	$B_1 = 8\times10^3$		0.90	6 837 275	13.16	0.807 039 42
	$D_2 = 314\,075$	$P_2 = 0.0609$	$B_2 = 5\times10^4$		0.80	6 837 275	13.16	0.807 039 42
	$D_3 = 136\,665$	$P_3 = 0.0265$			0.70	1 283 272	2.47	0.602 953 87
2000	$D_0 = 9\,766\,236$	$P_0 = 0.7175$	$B_0 = 1\times10^3$	51 978 751	0.95	23 762 962	45.71	0.989 995 83
	$D_1 = 2\,215\,629$	$P_1 = 0.1629$	$B_1 = 8\times10^3$		0.90	7 075 015	13.61	0.931 846 60
	$D_2 = 1\,106\,942$	$P_2 = 0.0813$	$B_2 = 5\times10^4$		0.80	7 075 015	13.61	0.931 846 60
	$D_3 = 521\,766$	$P_3 = 0.0383$			0.70	1 492 207	2.87	0.447 634 43

7 Conclusion

Based on the UTXO model of the Bitcoin system, we propose a pruning strategy and synchronization method to improve fast bootstrap service. Further pruning is carried out on the current system state UTXOs, and the snapshot synchronization strategy is improved to ensure the validity and verifiability of the pruned data. Experiments demonstrate that joining nodes can reduce the data download and storage by approximately 99.70% while saving approximately 97.23% of download time. At the latest blocks, the data volume decreases from approximately 435.57 GB to 1.33 GB, and the synchronization time decreases from approximately six hours to 10 minutes. Additionally, as the block height grows, the space savings may be even more considerable.

In future work, we plan to explore other effective pruning strategies that do not adhere to simple and brute-force methods. How to securely apply synchronization strategies to the Bitcoin system through velvet forks may also be investigated.

Acknowledgment

This work was supported by the National Key Project of China (No. 2020YFB1005700), the Natural Science Foundation of Shandong Province (No. ZR2021MF086), the National Key Research and Development Program of China (No. 2021YFA1000600), the National Natural Science Foundation of China (Nos. 62132018 and 62172117), the National Key Research and Development Program, the Young Scientist Scheme (No. 2022YFB3102400), and the National Key Research and Development Program of Guangdong Province (No. 2020B0101090002).

References

[1] G. W. Peters and E. Panayi, Understanding modern banking ledgers through blockchain technologies: Future of transaction processing and smart contracts on the internet of money, in *Banking Beyond Banks and Money*, P. Tasca, T. Aste, L. Pelizzon, and N. Perony, eds. Cham, Switzerland: Springer, 2016, pp. 239–278.

[2] G. W. Peters, E. Panayi, and A. Chapelle, Trends in crypto-currencies and blockchain technologies: A monetary theory and regulation perspective, arXiv preprint arXiv: 1508.04364, 2015.

[3] L. Wang, W. Liu, and X. Han, Blockchain-based government information resource sharing, in *Proc. IEEE 23rd Int. Conf. Parallel and Distributed Systems*, Shenzhen, China, 2017, pp. 804−809.

[4] X. Zhang and Y. Yin, Research on digital copyright management system based on blockchain technology, in *Proc. IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conf.*, Chengdu, China, 2019, pp. 2093–2097.

[5] R. Casado-Vara, F. de la Prieta, J. Prieto, and J. M. Corchado, Blockchain framework for IoT data quality via edge computing, in *Proc. 1st Workshop on Blockchain-Enabled Networked Sensor Systems*, Shenzhen, China, 2018, pp. 19–24.

[6] X. Wang, X. Zha, W. Ni, R. P. Liu, Y. J. Guo, X. Niu, and K. Zheng, Survey on blockchain for internet of things, *Comput. Commun.*, vol. 136, pp. 10–29, 2019.

[7] Y. Pu, T. Xiang, C. Hu, A. Alrawais, and H. Yan, An efficient blockchain-based privacy preserving scheme for vehicular social networks, *Inf. Sci.*, vol. 540, pp. 308–324, 2020.

[8] M. Mettler, Blockchain technology in healthcare: The revolution starts here, in *Proc. IEEE 18th Int. Conf. E-Health Networking, Applications and Services*, Munich, Germany, 2016, pp. 1–3.

[9] C. Li and B. Palanisamy, Incentivized blockchain-based social media platforms: A case study of *steemit*, in *Proc. 10th ACM Conf. Web Science*, Boston, MA, USA, 2019, pp. 145–154.

[10] W. Li, W. Meng, Y. Wang, and J. Li, Enhancing blackslist-based packet filtration using blockchain in wireless sensor networks, in *Proc. 16th Int. Conf. Wireless Algorithms, Systems, and Applications*, Nanjing, China, 2021, pp. 624–635.

[11] Y. Chen, H. Li, K. Li, and J. Zhang, An improved P2P file system scheme based on IPFS and blockchain, in *Proc. 2017 IEEE Int. Conf. Big Data*, Boston, MA, USA, 2017, pp. 2652–2657.

[12] A. S. Patil, R. Hamza, A. Hassan, N. Jiang, H. Yan, and J. Li, Efficient privacy-preserving authentication protocol using PUFs with blockchain smart contracts, *Comput. Secur.*, vol. 97, p. 101958, 2020.

[13] H. Yuan, X. Chen, J. Wang, J. Yuan, H. Yan, and W. Susilo, Blockchain-based public auditing and secure deduplication with fair arbitration, *Inf. Sci.*, vol. 541, pp. 409–425, 2020.

[14] Satoshi Nakamot, Bitcoin: A peer-to-peer electronic cash system. https://bitcoin. org/bitcoin.pdf, 2008.

[15] G. Wood, Ethereum: A secure decentralised generalised transaction ledger, *Ethereum Proj. Yellow Paper*, vol. 151, pp. 1–32, 2014.

[16] A. Kiayias, N. Leonardos, and D. Zindros, Mining in logarithmic space, in *Proc. 2021 ACM SIGSAC Conf. Computer and Communications Security*, Virtual Event, 2021, pp. 3487–3501.

[17] A. Chepurnoy, M. Larangeira, and A. Ojiganov, Rollerchain, a blockchain with safely pruneable full blocks, arXiv preprint arXiv: 1603.07926, 2016.

[18] R. Matzutt, B. Kalde, J. Pennekamp, A. Drichel, M.

Henze, and K. Wehrle, How to securely prune bitcoin's blockchain, in *Proc. 2020 IFIP Networking Conf.*, Paris, France, 2020, pp. 298–306.

[19] R. Matzutt, B. Kalde, J. Pennekamp, A. Drichel, M. Henze, and K. Wehrl, CoinPrune: Shrinking bitcoin's blockchain retrospectively, *IEEE Trans. Netw. Serv. Manage.*, vol. 18, no. 3, pp. 3064–3078, 2021.

[20] L. Ren, W. T. Chen, and P. A. S. Ward, SnapshotSave: Fast and low storage demand blockchain bootstrapping, in *Proc. 36th Annual ACM Symposium on Applied Computing*, Virtual Event, 2021, pp. 291–300.

[21] E. Palm, O. Schelén, and U. Bodin, Selective blockchain transaction pruning and state derivability, in *Proc. 2018 Crypto Valley Conf. Blockchain Technology*, Zug, Switzerland, 2018, pp. 31–40.

[22] B. Bünz, L. Kiffer, L. Luu, and M. Zamani, FlyClient: Super-light clients for cryptocurrencies, in *Proc. 2020 IEEE Symp. Security and Privacy*, San Francisco, CA, USA, 2020, pp. 928–946.

[23] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, A secure sharding protocol for open blockchains, in *Proc. 2016 ACM SIGSAC Conf. Computer and Communications Security*, Vienna, Austria, 2016, pp. 17–30.

[24] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, OmniLedger: A secure, scale-out, decentralized ledger via sharding, in *Proc. 2018 IEEE Symp. Security and Privacy*, San Francisco, CA, USA, 2018, pp. 583–598.

[25] M. Zamani, M. Movahedi, and M. Raykova, RapidChain: Scaling blockchain via full sharding, in *Proc. 2018 ACM SIGSAC Conf. Computer and Communications Security*, Toronto, Canada, 2018, pp. 931–948.

[26] J. Wang and H. Wang, Monoxide: Scale out blockchain with asynchronous consensus zones, in *Proc. 16th USENIX Conf. Networked Systems Design and Implementation*, Boston, MA, USA, 2019, pp. 95–112.

[27] M. J. Amiri, D. Agrawal, and A. El Abbadi, SharPer: Sharding permissioned blockchains over network clusters, in *Proc. 2021 Int. Conf. Management of Data*, Xi'an, China, 2021, pp. 76–88.

[28] Z. Hong, S. Guo, P. Li, and W. Chen, Pyramid: A layered sharding blockchain system, in *Proc. IEEE INFOCOM 2021-IEEE Conf. Computer Communications*, Vancouver, Canada, 2021, pp. 1–10.

[29] H. Yu, I. Nikolić, R. Hou, and P. Saxena, OHIE: Blockchain scaling made simple, in *Proc. 2020 IEEE Symp. Security and Privacy*, San Francisco, CA, USA, 2020, pp. 90–105.

[30] J. Hellings and M. Sadoghi, ByShard: Sharding in a byzantine environment, *Proc. VLDB Endow.*, vol. 14, no. 11, pp. 2230–2243, 2021.

[31] S. Li, M. Yu, C. S. Yang, A. S. Avestimehr, S. Kannan, and P. Viswanath, PolyShard: Coded sharding achieves linearly scaling efficiency and security simultaneously, *IEEE Trans. Inform. Forensics Secur.*, vol. 16, pp. 249–261, 2021.

[32] D. Perard, J. Lacan, Y. Bachy, and J. Detchart, Erasure code-based low storage blockchain node, in *Proc. 2018 IEEE Int. Conf. Internet of Things* (*iThings*) *and IEEE Green Computing and Communications* (*GreenCom*) *and IEEE Cyber, Physical and Social Computing* (*CPSCom*) *and IEEE Smart Data* (*SmartData*), Halifax, Canada, 2018, pp. 1622–1627.

[33] D. Mitra and L. Dolecek, Patterned erasure correcting codes for low storage-overhead blockchain systems, in *Proc. 2019 53rd Asilomar Conf. Signals, Systems, and Computers*, Pacific Grove, CA, USA, 2019, pp. 1734–1738.

[34] B. Bitcoin Project, Bitcoin core version 0.11.0 released, https://github.com/bitcoin/bitcoin/blob/master/doc/release-notes/release-notes-0.11.0.md, 2015.

[35] X. Feng, J. Ma, Y. Miao, Q. Meng, X. Liu, Q. Jiang, and H. Li, Pruneable sharding-based blockchain protocol, *Peer-to-Peer Netw. Appl.*, vol. 12, no. 4, pp. 934–950, 2019.

[36] H. Schoenfeld and A. Molina, Pascal: An infinitely scalable cryptocurrency, https://www.pascalcoin.org/storage/whitepapers/PascalWhitePaperV5.pdf, 2019.

[37] B. S. Reddy, securePrune: Secure block pruning in UTXO based blockchains using accumulators, in *Proc. 2021 Int. Conf. COMmunication Systems & NETworkS*, Bangalore, India, 2021, pp. 174–178.

[38] J. Cai, K. Qian, J. Luo, and K. Zhu, SARM: Service function chain active reconfiguration mechanism based on load and demand prediction, *Int. J. Intell. Syst.*, vol. 37, no. 9, pp. 6388–6414, 2022.

[39] J. Cai, H. Fu, and Y. Liu, Deep reinforcement learning-based multitask hybrid computing offloading for multiaccess edge computing, *Int. J. Intell. Syst.*, vol. 37, no. 9, pp. 6221–6243, 2022.

[40] W. Li, Y. Wang, Z. Jin, K. Yu, J. Li, and Y. Xiang, Challenge-based collaborative intrusion detection in software-defined networking: An evaluation, *Digital Commun. Netw.*, vol. 7, no. 2, pp. 257–263, 2021.

[41] T. Li, W. Chen, Y. Tang, and H. Yan, A homomorphic network coding signature scheme for multiple sources and its application in IoT, *Secur. Commun. Netw.*, vol. 2018, p. 9641273, 2018.

[42] Q. Chen, C. Tang, and Z. Lin, Efficient explicit constructions of multipartite secret sharing schemes, *IEEE Trans. Inform. Theory*, vol. 68, no. 1, pp. 601–631, 2022.

[43] Q. Chen, C. Tang, and Z. Lin, Compartmented secret sharing schemes and locally repairable codes, *IEEE Trans. Commun.*, vol. 68, no. 10, pp. 5976–5987, 2020.

[44] Q. Chen, C. Tang, and Z. Lin, Efficient explicit constructions of compartmented secret sharing schemes, *Des. Codes Cryptogr.*, vol. 87, no. 12, pp. 2913–2940, 2019.

[45] K. Mo, T. Huang, and X. Xiang, Querying little is enough: Model inversion attack via latent information, in *Proc. 3rd Int. Conf. on Machine Learning for Cyber Security*,

Guangzhou, China, 2020, pp. 583–591.

[46] K. Mo, W. Tang, J. Li, and X. Yuan, Attacking deep reinforcement learning with decoupled adversarial policy, *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 1, pp. 758–768, 2023.

[47] W. X. Liu, J. Cai, Q. C. Chen, and Y. Wang, DRL-R: Deep reinforcement learning approach for intelligent routing in software-defined data-center networks, *J. Netw. Comput. Appl.*, vol. 177, p. 102865, 2021.

[48] F. Wang, Y. Li, F. Liao, and H. Yan, An ensemble learning based prediction strategy for dynamic multi-objective optimization, *Appl. Soft Comput.*, vol. 96, p. 106592, 2020.

[49] L. Hu, H. Yan, L. Li, Z. Pan, X. Liu, and Z. Zhang, MHAT: An efficient model-heterogenous aggregation training scheme for federated learning, *Inf. Sci.*, vol. 560, pp. 493–503, 2021.

[50] W. Tang, B. Li, M. Barni, J. Li, and J. Huang, An automatic cost learning framework for image steganography using deep reinforcement learning, *IEEE Trans. Inform. Forensics Secur.*, vol. 16, pp. 952–967, 2021.

Arthur Sandor Voundi Koe received the BEng degree in network and computer maintenance from The African Institute of Computer Science, Cameroon in 2010, the second BEng degree in fundamental computer science from University of Yaoundé 1, Cameroon in 2011, the MEng degree in computer and application technology from Hunan University, China in 2015, and the PhD degree in computer science and application technology from Hunan University, China in 2020. He is currently a doctoral researcher at Guangzhou University, China. His research interests include security and privacy issues in blockchain technology, cloud computing security and privacy issues, and machine learning.

Duncan S Wong received the BEng degree in electrical & electronic engineering with first-class honors from The University of Hong Kong, China in 1994, the M. Phil. degree in information engineering from The Chinese University of Hong Kong in 1998, and the PhD degree in computer science from Northeastern University, Boston, MA, USA in 2002. He is currently an off-campus supervisor at Guangzhou University. His main research interest is cryptography, particularly, cryptographic protocols, encryption, and signature schemes, and anonymous systems.

Hai Jiang received the BEng and MEng degrees from Jilin University in computer science and technology, China, and the PhD degree from Institute of Computing Technology, Chinese Academy of Sciences, China. He is currently an off-campus supervisor at Guangzhou University, China. He has published more than 10 academic papers. His research interests include blockchain and distributed computing networks.

Pengfei Huang received the BEng degree in computer science and technology from Harbin Engineering University, China in 2019. He is currently a master student in cyberspace security at Guangzhou University, China. His main research interest is blockchain.

Xiaojun Ren received the BEng and MEng degrees from Shandong University of Science and Technology, China in 2007 and 2010, respectively, and the PhD degree from Dongseo University, Republic of Korea in 2016. Currently, he is a full-time doctoral researcher at Institute of Artificial Intelligence and Blockchain, Guangzhou University, China. His research interests include blockchain and machine learning.

Teng Huang received the BEng degree from Guilin University of Technology, China in 2009, the MEng degree from Central South University, China in 2012, and the PhD degree in electronic and communication engineering from Beihang University, China in 2019. He is currently an associate professor at Institute of Artificial Intelligence and Blockchain, Guangzhou University, China. His research focuses on machine learning, adverserial attacks, and synthetic aperture radar target recognition.