Front. Comput. Sci., 2023, 17(3): 173403
https://doi.org/10.1007/s11704-022-1685-5

RESEARCH ARTICLE

Gauze: enabling communication-friendly block synchronization with cuckoo filter

Xiaoqiang DING[1], Liushun ZHAO[2], Lailong LUO[3], Junjie XIE[4], Deke GUO (✉)[1,3], Jinxi LI[1]

1 College of Intelligence and Computing, Tianjin University, Tianjin 300350, China
2 School of Computer Science and Technology, Xidian University, Xi'an 710071, China
3 Science and Technology on Information Systems Engineering Laboratory, National University of Defense Technology, Changsha 410073, China
4 Institute of Systems Engineering, AMS, PLA, Beijing 100141, China

Abstract Block synchronization is an essential component of blockchain systems. Traditionally, blockchain systems tend to send all the transactions from one node to another for synchronization. However, such a method may lead to an extremely high network bandwidth overhead and significant transmission latency. It is crucial to speed up such a block synchronization process and save bandwidth consumption. A feasible solution is to reduce the amount of data transmission in the block synchronization process between any pair of peers. However, existing methods based on the Bloom filter or its variants still suffer from multiple roundtrips of communications and significant synchronization delay. In this paper, we propose a novel protocol named Gauze for fast block synchronization. It utilizes the Cuckoo filter (CF) to discern the transactions in the receiver's mempool and the block to verify, providing an efficient solution to the problem of set reconciliation in the P2P (Peer-to-Peer Network) network. By up to two rounds of exchanging and querying the CFs, the sending node can acknowledge whether the transactions in a block are contained by the receiver's mempool or not. Based on this message, the sender only needs to transfer the missed transactions to the receiver, which speeds up the block synchronization and saves precious bandwidth resources. The evaluation results show that Gauze outperforms existing methods in terms of the average processing latency (about 10× lower than Graphene) and the total synchronization space cost (about 10× lower than Compact Blocks) in different scenarios.

Keywords block synchronization, cuckoo filter, probabilistic data structure

1 Introduction

The blockchain is a distributed system that combines cryptography and P2P networks, which enables transactions to be completed without a third party and protects the users' privacy. It is a distributed ledger that records transactions in the form of an append-only ledger maintained by all network participants. Furthermore, the recorded transactions are protected by encryption technologies and stored in blocks connected in a chain structure. There are two critical roles in the above process of recording the ledger: i) miners collect transactions, verify their integrity, pack them into a new block, and append the block to the blockchain; ii) the non-mining peers add the new block to their local blockchain after successful verification.

When a node in the blockchain generates a new transaction with an encrypted hashed transaction ID, the new transaction should be broadcasted to all peers in the network using the Gossip protocol [1]. Thus, the higher the number of transactions, the longer the synchronization time required. The peers who received the new transaction then keep it in their local mempool for validation. Subsequently, the miners are engaged in a time-sensitive competition to solve a Proof-of-Work problem [2], or a Proof-of-Stake problem [3,4] to extend the blockchain. After the succeeded miner packs the transactions from the mempool as a new block, the new block will be broadcasted to other peers across the network (illustrated in Fig. 1). The above process of broadcasting these transactions to each peer and ensuring the absolute consistency of the transactions contained in the blocks is called block synchronization. There are two main goals of block synchronization: first, allowing network peers to synchronize the update of transactions and blocks of the system; second, disseminating peer information to support peers to re-enter the system after disconnection.

Block synchronization is of great importance for blockchain systems. Network delays and overhead are noticeable during block synchronization. It is crucial for blockchain systems to achieve a high level of processing performance while ensuring the consistency of the block data. There are many advantages for an efficient protocol to synchronize new authorized transactions and newly mined blocks of validated transactions among peers. First, reducing the propagation time of the

Received November 26, 2021; accepted March 17, 2022

E-mail: guodeke@gmail.com

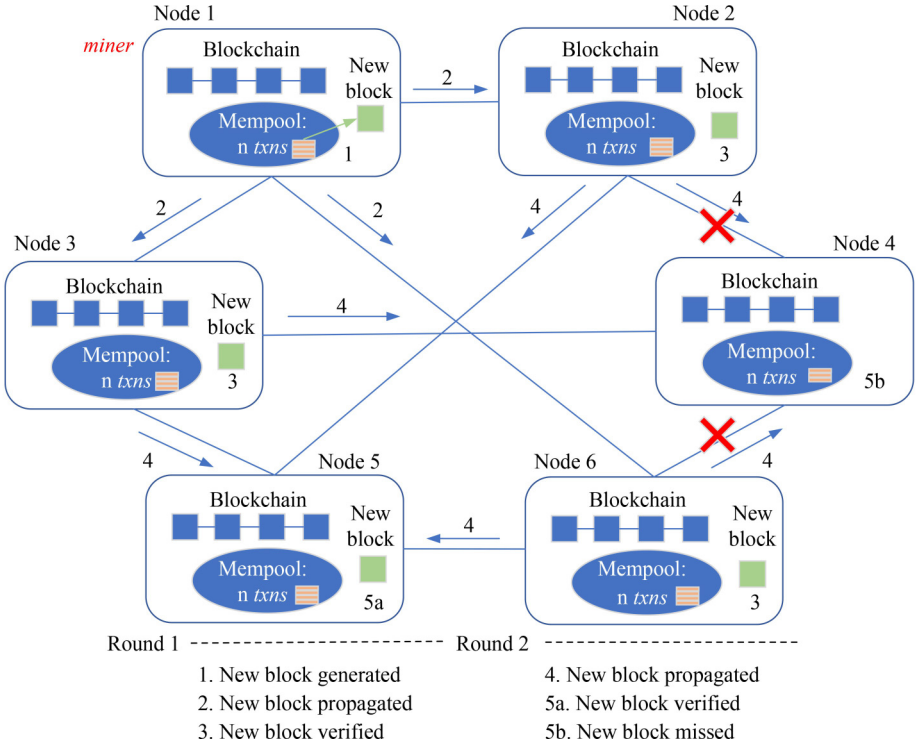

Fig. 1 Block synchronization in the blockchain. (Since node 4 is disconnected, new transactions and blocks are not saved to this node's mempool during the second synchronization round.)

underlying blockchain network is crucial for miners, as every millisecond of delay would reduce the chances of gaining bookkeeping rights. Second, less data transmission can reduce the network communication burden, thus improving the process efficiency. Such a fast and efficient block synchronization process is an essential basis for ensuring the performance expansion of the blockchain. However, there are still several intractable challenges in the synchronization process.

First, how to reduce the bandwidth consumption during block synchronization [5,6] is a vital issue for blockchain systems. Traditionally, the same transaction may be transmitted to other peers twice in the block synchronization process. The first transmission is sending the newly authorized transactions to all peers, and the second transmission is the broadcast across the network as a content of the newly mined blocks. However, the majority of the peers have already received these transactions, and they only need to discern them from those in their mempool. The transmission of these repeated transaction data requires high network bandwidth resources.

Second, the propagation delay for distributing new blocks is still very high. How to improve the processing performance and speed up the block synchronization process [7,8], which will directly affect the transaction throughput. The overall throughput of current blockchain systems is generally low. Specifically, the transactions per second (tps) of Bitcoin and Ethereum are 7 and 15, respectively. The tps of blockchains are much lower than that of centralized payment systems such as VISA (24,000 tps), making it difficult to meet the practical requirements of users. If the peers can complete the block synchronization process quickly and efficiently, they can start the next round of competition for the right to keep track of the blocks as early as possible.

However, existing methods either take excessive time or require high network bandwidth [9−12] to complete block synchronization. Moreover, some methods may also require several roundtrips to complete the block synchronization, which further degrades the processing performance of the blockchain systems. Xtreme Thinblocks [13] (XThin) uses a Bloom filter [14] to encode the transaction IDs. The sender responds to the receiver with a list of the block's transaction IDs shortened to only 8-bytes. However, the space cost of XThin is still very high. Compact Blocks [15] (CB) reduces the network bandwidth consumption to $6n$ (n is the number of transactions encoded in a block) by shortening the block's transaction IDs to 6-bytes. Graphene [16] further reduce the network bandwidth consumption through the combination of a Bloom filter and an Invertible Bloom Lookup Table (IBLT) [17]. However, Compact Blocks and Graphene may require an extra roundtrip time when the receiver misses transactions.

Besides, the existing methods are not friendly to the performance and efficiency of blockchains. Therefore, a significant improvement could be achieved if a fast method could be proposed to synchronize block transactions. The majority of the peers have already received these transactions. Instead of exchanging all transactions in the blocks directly between peers, we aim to discern them from those in their mempool and only transmit the missed transactions between peers.

In this paper, we introduce Gauze, a probabilistic method for synchronizing blocks (and mempools) among peers in blockchains with high success probability. The basic idea of Gauze is to use the Cuckoo filter [18] to discern the different

transactions between peers. By exchanging the CFs, Gauze can determine those transactions that existed in the target node and avoid re-transmission in the subsequent synchronization processes. Our work can save a significant portion of network bandwidth by reducing transmission traffic in the network. Our Gauze provides higher process performance, uses less space than other approaches, and does not require an extra roundtrip time to accomplish the block synchronization process. The main contributions of this paper are summarized as follows:

1) For the most common scenario where the receiver's mempool contains all transactions in the block, we design the Gauze protocol. Gauze uses the sender's CF to determine the transactions in a block, which can speed up the transaction synchronization process among peers and improve the overall processing performance of the blockchain.

2) For the missed transactions owing to network failures or other reasons during the block synchronization, we design the Gauze Enhanced protocol. Gauze Enhanced confirms the lost transactions and transmits them to the receiver directly. It can reliably realize block synchronization in more complex scenarios and efficiently reduce the network bandwidth consumption for block synchronization during block propagation.

3) The evaluation results show that our protocols require less data transmission (only about one-tenth of Compact Blocks [15] and one-fifth of Graphene [16] when the block size is 200) and achieve higher processing performance (increased by about 10× when the block size is 200, and about 3× when the block size is 10,000) in blockchain systems.

The rest of this paper is organized as follows. Section 2 introduces the background and related work. Section 3 details the design philosophy of our Gauze protocol. Section 4 reports the evaluation results. Section 5 discusses several issues about our Gauze protocol and at last Section 6 concludes the whole paper.

2 Background and related work

In this section, we first introduce the background of block synchronization and follow with its related studies. Then we introduce the data structures of a set reconciliation.

2.1 Block synchronization

Block synchronization (illustrated in Fig. 1) is an essential function of blockchain systems that provides the necessary conditions for achieving a consensus among miners. Block synchronization includes the synchronization of the transactions and the state. Transaction synchronization is aimed to ensure that transactions of the blockchain reach as many peers as possible. This can alleviate the impact of transaction loss and provide the basis for packaging transactions into blocks through consensus. The function of state synchronization tries to keep the state of the blockchain peers up-to-date and ensure the disconnected peers are correctly returned to the latest state. It is noteworthy that only a node that holds the latest block state can participate in the consensus. As shown in Fig. 1, when node 1 mines a new block and broadcasts it to other nodes, node 4 is disconnected for a while due to network failure or other reasons, and the new transactions and block are not saved to the node's mempool. Therefore, node 4 needs transaction and state synchronization operations to recover to the latest block state.

Blocks propagate in a similar way (illustrated in Fig. 2). When new transactions are received, a node sends the transaction IDs as the content of an INVENTORY (*inv*) message (the first step in Fig. 2), which is usually the header of the block, to each neighbor. If some neighbors do not have all of these transactions, they will send a getdata message to request the missed transactions (the second step in Fig. 2). After receiving the getdata message from the receiver, the sender sends the block header of missed transactions to the receiver (the third step in Fig. 2). Block synchronization ensures transactions in each block of blockchain kept in each node are identical. There are two methods of block synchronization. In the first method, the sender would transmit the complete block, and then all transactions in the block are sent to the receiver. This undoubtedly leads to significant bandwidth consumption. For the second method, each node only sends the transactions that do not exist at the receiver, which is explored in this paper. This approach would reduce the bandwidth consumption and improve the block synchronization processing performance against the first method.

2.2 Comparison to related work

Xtreme Thinblocks [13] (XThin) is a robust and efficient protocol designed to accelerate the block relaying. It uses the Bloom filter to send only the missed transaction of the receiver. Further, it shortens the block's transaction IDs to 8-bytes, which reduces the size of the XThin blocks. The cost of XThin is approximately $\frac{m\log_2(f)}{8\ln 2}+6n$ (n is the number of transactions encoded in a block) bytes. In addition, Xthinner [19] is a variant of XThin that uses compression techniques for the list of transactions in a block. In comparison, Gauze requires a significantly lower bandwidth consumption.

For the Compact Blocks [15], the getdata message from the receiver is a simple request (no Bloom filter is sent). The sender shorts the block's transaction IDs to 6-bytes. Although the network cost of the Compact Blocks is $6n$ (n is the number of transactions encoded in a block), it requires an extra roundtrip time when the receiver misses some transactions. As shown in Section 4, Gauze outperforms Compact Blocks significantly in terms of space cost.

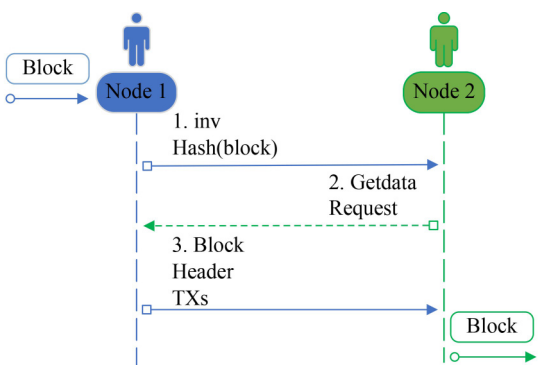

Fig. 2 The propagation process of blocks in the blockchain

Graphene [16] further reduces the network bandwidth consumption through the combination of a Bloom filter and an invertible IBLT. It uses a Bloom filter to identify the symmetric difference between the block and mempool, and the IBLT is employed to recover from minor errors in the Bloom filter. However, Graphene requires that the generated BF and IBLT at the sender and receiver sides be nearly the same size to support the subtraction operation. In addition, Graphene may use an additional roundtrip to repair missed transactions compared to Gauze.

Erlay [20] is a hybrid protocol that combines low-fanout flooding with efficient set reconciliation. It is designed to optimize Bitcoin's transaction relay while maintaining the existing security guarantees. Erlay is more scalable relative to the number of peers in the network and their connectivity, reducing the bandwidth used to announce transactions by 84%.

2.3 Set Reconciliation data structures

Cuckoo filter (CF) [18,21] is a lightweight probabilistic data structure, which supports constant-time membership queries, dynamically element addition, and item removals. A CF consists of k buckets, each of which consists of b entities, and each entity contains an f-bit fingerprint. CF provides two candidate insertion positions for each element and randomly selects one of the two empty positions for element insertion. For an element x, if both positions are non-empty, CF randomly selects a candidate position and kicks out the element stored in this position to accommodate x's fingerprint η_x. The kicked-out element would then be inserted into its alternative candidate position. If there is still an element in this alternative position, the kick-out process will continue, and the process ends until the element is inserted successfully. Otherwise, the insertion will fail when the number of kick-out operations reaches a given threshold, *MaxLoop*. Here, *MaxLoop* is a parameter that prevents duplicate insertion entirely, at the cost of slowing down insertion if the table already contains a (false positive) matching entry for the bucket and fingerprint.

CF utilizes a technique called partial-key cuckoo hashing [22] to derive an alternative location of an item based on its fingerprint. Element x is associated with an f-bit fingerprint $\eta_x = hash_0(x) \ mod \ 2^f$. For this item x, the cuckoo hashing scheme calculates the indexes of the two candidates as follows:

$$h_1(x) = hash(x), \tag{1}$$

$$h_2(x) = h_1(x) \oplus hash(x'sfingerprint). \tag{2}$$

There are many other data structures for set reconciliation [23], including the Counting Bloom filter (CBF) [24], Invertible Bloom filter (IBF) [25], IBLT [17], Marked Cuckoo Filter (MCF) [26], and Merkle tree [27]. The sender encodes the data in a CBF, IBF, or IBLT, and the receiver eliminates the common elements by subtracting filters. The nodes in the Merkle tree are compared as identical pruned subtrees to derive the different elements between the two sets. CF does not need to support subtracting filters or pruning subtrees operations compared to these data structures. Moreover, CF supports adding and removing items dynamically while achieving higher performance than others.

3 The Gauze protocol

The block synchronization process of the entire network node can be regarded as the synchronization between multiple peers of the sender and receiver. Existing methods either take excessive time or require higher network bandwidth resources to complete the block synchronization. Some methods even require additional roundtrips in the block synchronization process. Undoubtedly, these weaknesses further degrade the performance and efficiency of blockchain systems. After a comprehensive analysis and consideration, we decide to use the Cuckoo filter to conduct the set reconciliation of the blocks and mempools. CF has many advantages and can well meet the requirements of block synchronization. First, CF supports adding and removing items dynamically and provides higher lookup performance than the others. Second, CF is more accessible to implement than alternatives such as IBLTs and causes lower spatial and other costs than a Bloom filter or IBLTs. Based on these considerations, we propose Gauze. The goal of the Gauze protocol is to reduce the network bandwidth consumption and improve the processing performance during the block synchronization process. In this section, we discuss how CF can be used for synchronizing blocks and mempools so as to reduce the amount of resultant network traffic.

3.1 Problem definition

In this paper, we define two scenarios of applying Gauze to solve the block synchronization problem during block propagation. The two scenarios together can cover almost all block synchronization cases in the blockchain.

For the first scenario, we assume that the transactions in the block are a subset of the receiver's mempool (m transactions in mempool and n transactions in block). As shown in Fig. 3, all transactions in the block are contained in the receiver's mempool. This is the most common scenario in blockchains. We designed Protocol 1 for this scenario, which is described in Section 3.2.

For the second scenario, we assume that the mempool of the receiver does not contain the entire block (m transactions in mempool and y transactions in a block and mempool), as shown in Fig. 4. Some transactions of the block (n transactions in block) are not contained in the mempool owing to network failures or other reasons (red area in Fig. 4). For this scenario, Protocol 2 would play a vital role instead of Protocol 1. Protocol 2 is a supplement to protocol 1, it uses two CFs to

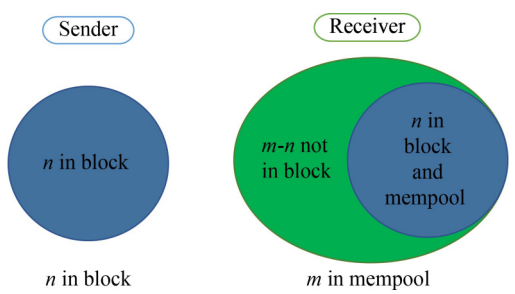

Fig. 3 Scenario 1: the receiver's mempool contains the entire block

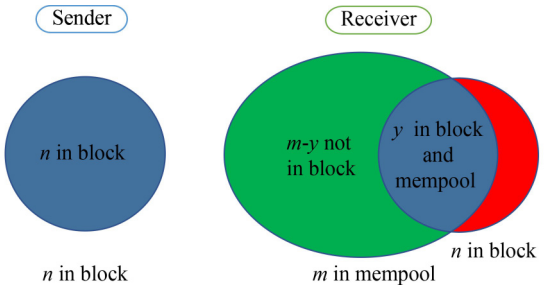

Fig. 4 Scenario 2: the receiver's mempool does not contain the entire block

determine the missed transactions and transmit the missed transactions to the receiver directly. Besides, Protocol 2 is significantly efficient because it takes only one roundtrip to achieve block synchronization.

The main block synchronization steps of Gauze are listed as follows:

1) Protocol 1 is invoked to conduct the block synchronization.

2) If during the query in the receiver's mempool, the CF_S will still not be empty after removing all discovered transactions from the CF_S, we know that Protocol 1 fails and start Protocol 2.

3.2 Gauze protocol

We design Protocol 1 (as illustrated in Fig. 5), for the first scenario, where the receiver's mempool contains all transactions in the block (Fig. 3-right). The key idea of Protocol 1 is to use the sender's CF to determine which transactions of the receiver need to be packaged into the block. Protocol 1 can solve most of the block synchronization problems in blockchains. Protocol 1 is composed of the following steps:

1) The sender encodes the transactions of the block as CF_S and sends it to the receiver.

2) The receiver queries the CF_S received from the sender to identify which transactions in the mempool are included in the block. Then she adds these transactions to S_1 and packages these transactions into its own block.

After these two steps, the receiver can confirm the transactions that need to be packaged into the block. The sender does not transmit additional content, just the CF_S to the receiver. This significantly reduces the network bandwidth consumption and improves the processing performance and efficiency of the block synchronization.

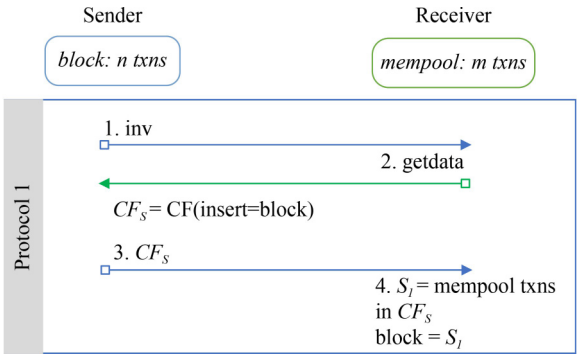

Fig. 5 An illustrative example of Protocol 1 for propagating a block that is a subset of the mempool

Protocol 1 Gauze

1: Sender: The sender sends the transaction IDs as the content of an inventory (*inv*) for the block to the receiver.
2: Receiver: The receiver requests the unknown block.
3: Sender: The sender creates CF_S from the transaction IDs of the block (blue area in Fig. 3-left) and sends it to the receiver.
4: Receiver: The receiver creates a candidate set S_1 of transaction IDs, queries the CF_S received from the sender, and adds the transactions contained in the CF_S to S_1 from her mempool. Based on the result of S_1, she validates the Merkle root in the block header and decodes the block.

3.3 Gauze enhanced protocol

Protocol 1 will not succeed if the receiver does not have all transactions in the block. In this situation, the receiver would invoke Protocol 2, which is illustrated in Fig. 6. Protocol 2 is designed for the second scenario when the receiver's mempool does not contain all transactions in the block (Fig. 4-right). Protocol 2 uses two CFs to confirm the lost transactions and transmit them to the receiver. The main steps of Protocol 2 are listed as follows:

1) The receiver encodes the transactions of the mempool as CF_R and transmits it to the sender.

2) The sender uses the CF_R to query the missed transactions of the receiver, adds the transactions to the **miss_txn**, and then sends **miss_txn** to the receiver.

3) The receiver uses the sender's CF_S to query which transactions in the mempool are contained in the block and then packs those transactions with the transactions in **miss_txn** into its block.

After these steps, the missed transactions in the receiver's mempool can be confirmed and transmitted to the receiver so as to achieve block synchronization. Using CF_S and **miss_txn** transmitted from the sender, the receiver can confirm the transactions in the block and thereafter package them into its

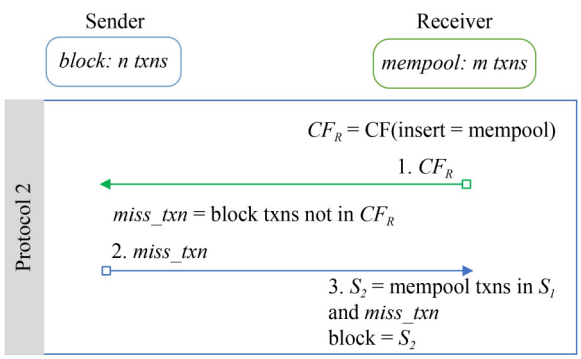

Fig. 6 If Protocol 1 fails (the block is not a subset of the mempool), Protocol 2 begins

Protocol 2 Gauze Enhanced

1: Receiver: The receiver creates CF_R from the transaction IDs of the mempool (blue and green areas in Fig. 4-right) and sends it to the sender.
2: Sender: The sender queries CF_R and confirms the receiver's missing transactions (red area in Fig. 4-right) according to the transactions contained in its block and adds them to the **miss_txn**.
3: Sender: The sender sends **miss_txn** containing all transactions that are not in CF_R directly to the receiver.
4: Receiver: The receiver creates a candidate set S_2 of transaction IDs that pass through CF_S sent by the sender in Protocol 1. Based on the results of S_2 and **miss_txn**, she validates the Merkle root in the block header and decodes the block.

own block. The receiver then validates the Merkle root in the block header and decodes the block. Protocol 2 uses another CF from the receiver to help the sender transmit the missed transactions to the receiver. It can cover the more complicated scenario in blockchain systems compared to Protocol 1.

3.4 Theoretical analysis

Gauze first uses Protocol 1 to realize block synchronization, which is the most common scenario in blockchain systems. Therefore, most peers are successfully synchronized during this process. For the second scenario, Protocol 2 is invoked. By exchanging the CF, the lost transaction can be identified, and then directly transmitted to complete the block synchronization in the subsequent transmission. The time complexity of insertion operation with high probability is $O(1)$ if the load is well managed. And the time complexity of the lookup operation is also $O(1)$ because it does not depend on the input size. Thus, the time complexity of Gauze is $O(n)$, which mainly occurs in the traversal of the transactions.

There are some parameter bounds set on CF, and the detailed proof can be referenced in literatures [18,28]. We introduce several vital parameters and briefly analyze them in this subsection.

The space cost C of block synchronization If each fingerprint is f bits and the hash table has a load factor of α, then the amortized space cost C of block synchronization for each item is

$$C = \frac{t}{n} = \frac{fe}{\alpha e} = \frac{f}{\alpha} bits, \tag{3}$$

where t is the table size, n is the number of items and e is the number of entries.

Upper bound of the false fingerprint The upper bound of the total probability of a false fingerprint hit is

$$1 - \left(1 - \frac{1}{2^f}\right)^{2b} \approx \frac{2b}{2^f}, \tag{4}$$

which is related to the bucket size b.

Minimal fingerprint size To obtain the target decoding rate during the block synchronization process, the false positive rate ε must be within a reasonable boundary. Therefore, the filter must guarantee that $2b/2^f \leqslant \varepsilon$. Thus, the minimum fingerprint length is approximately

$$f \geqslant \lceil \log_2(2b/\varepsilon) \rceil = \lceil \log_2(1/\varepsilon) + \log_2(2b) \rceil. \tag{5}$$

Upper Bound of total synchronization cost As shown in Eqs. (3) and (5), both f and α depend on the bucket size b. The average total synchronization cost C of block synchronization is bound by the following:

$$C \leqslant \lceil \log_2(1/\varepsilon) + \log_2(2b) \rceil / \alpha, \tag{6}$$

where ε is the target false positive rate, b is the number of entries per bucket and α ($0 \leqslant \alpha \leqslant 1$) is the load factor that increases with b. The space cost of cuckoo filters are asymptotically $1.05\log_2(1/\varepsilon)$ when $b = 4$ and $1/\alpha \approx 1.05$. However, Bloom filters require $\frac{1}{\ln 2}\log(1/\varepsilon) \approx 1.44\log_2(1/\varepsilon)$ bits or more for each item, for a 44% overhead compared to CF.

Decoding rate Given a target false positive rate ε, the overall decoding rate Δ_o can be calculated as follows:

$$\Delta_o = 1 - (1 - \varepsilon)^{(n_b - n_{mb})}, \tag{7}$$

where n_b is the number of transactions in the block, and n_{mb} is the number of transactions in the receiver's mempool included in the block. If the exact number of queries (x) is given, then the decoding rate is

$$\Delta_o(X = x) = C^x_{(n_b - n_{mb})} \varepsilon (1 - \varepsilon)^{(n_b - n_{mb} - x)}. \tag{8}$$

4 Evaluation

In this section, we compare Gauze with the Compact Blocks and Graphene methods from several aspects. The primary purpose of our experiment is to measure the parameters of network data exchanged by peers during the block synchronization process. We mainly consider the local processing latency required to complete block synchronization. This is because improving the local processing efficiency of peers and reducing the processing latency can improve the efficiency of block synchronization. We describe the experiment settings and the evaluation results of our protocols' performance and network cost to indicate the methods.

4.1 Experimental settings

The main parameters of our experiments are set as follows:

a) Block size and transaction IDs In all experiments, we mainly evaluate three different block sizes in terms of transactions: 200, 2,000, and 10,000. Note that the average block size of Bitcoin cash (BCH) and Ethereum (ETH) blocks is 200, and the average size of Bitcoin (BTC) blocks is 2,000. We set 10,000 to cope with the possible expansion of the blockchain in the future and use this size to simulate block synchronization in large-scale scenarios. The transaction ID of each transaction occupies 8-bytes in our experiments.

b) Cuckoo filter A Cuckoo filter consists of k buckets, each of which consists of b entities and each entity contains an f-bits fingerprint. In our experiments, we set k = block size, b = 4, and f = 8 for the Cuckoo filter because these settings were evaluated as the best in literature [18].

c) Performance metrics To fully compare the differences between the Gauze, Compact Blocks and Graphene protocols, we compare them using the following metrics:

1) Local processing latency: We evaluate the processing performance with Local processing latency. The latency that peers process transactions locally and completes the synchronization process.

2) Total synchronization cost: The amount of data transferred during the block synchronization.

3) False positive rate: Measured by querying a filter with non-existing transactions and counting the fraction of positive returns.

4.2 Evaluation of Gauze protocol

In this subsection, we aim to validate Protocol 1 of Gauze designed for the first scenario. In this scenario (Fig. 3), the transactions in the receiver's mempool consist of all transactions in the block (expressed as n) plus some additional

transactions not contained in the block (expressed as $m-n$), which increase along the x-axis as a multiple of the block size. Here, multiple is the ratio of $\frac{m-n}{n}$, where m and n are the number of transactions in mempool and block, respectively, and $m-n$ is the number of transactions not in the block. We change the mempool size from 0 to 5.5 times according to the block size to evaluate the trend of the three performance standards as the transaction volume increases. For example, if a block holds $n=200$ transactions and the multiple is 2.0. The receiver's mempool would consist of $m=600$ transactions, 200 transactions in the block, plus $2.0\times200=400$ additional transactions (illustrated in Fig. 3).

Local processing latency Figure 7 shows the local processing latency of Gauze and Graphene as the mempool size increases. The processing latency of Graphene is approximately 10× greater than that of Gauze when the block size is 200 and is approximately 4× and 3× than that of Gauze when the block size is 2,000 and 10,000, respectively. As mentioned in Section 4, a Cuckoo filter just needs to check two locations for answering the query, so this operation is $O(1)$. However, Bloom filter or its variants with h hash functions, requiring $O(h)$. The cuckoo filter provides higher lookup performance than Bloom filters and IBLTs in some practical applications. These experiments prove that Gauze is better than Graphene in terms of the local processing performance, thus dealing with challenge 2.

Total synchronization cost Figure 8 shows the average space cost of each block in bytes of Gauze compared to Compact Blocks and Graphene, as the mempool size increases. The results verify that Gauze is superior to graphene and compact blocks in terms of synchronization cost. When the block size is 200, the synchronization cost of Gauze is only approximately one-tenth of that of compact blocks and one-fifth of that of graphene. Moreover, when the block size is 2,000 and 10,000, the synchronization cost is approximately one-eighth of compact blocks. In these two cases, the size of each Gauze block is also smaller than that of the Graphene block. As mentioned in Section 4, the average synchronization cost C is bound by Eq. (6) and approaches to $\log_2(1/\varepsilon)$, when ε varies from 0.001% to 10%. Equation (6) shows that cuckoo filters are asymptotically better (by a constant factor) than Bloom filters, which require $\frac{1}{\ln 2}\log(1/\varepsilon)\approx1.44\log_2(1/\varepsilon)$ bits or more for each item, for a 44% overhead compared to CF. These experiments verify that Gauze could significantly minimize the space cost required for synchronization among replicas of widely propagated information, thus dealing with challenge 1.

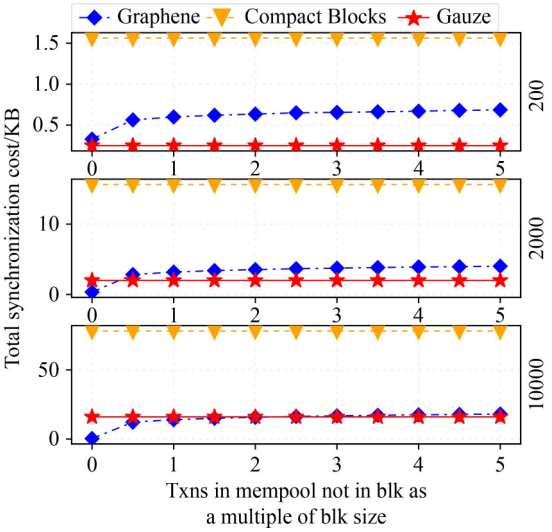

Fig. 8 Average space cost of Gauze VS. Compact Blocks and Graphene. Each facet is a block size: (200, 2000, and 10000 transactions)

False positive rate Figure 9 shows the false positive rates of Gauze compared to Graphene, as the mempool size increases. Gauze does not always achieve a lower false positive rate than Graphene. There is an inflection point in the graph of the Graphene. This is because when the *multiple* is 0, the transactions in the receiver's mempool and the transactions in the block are the same, and thus the number of false positive is

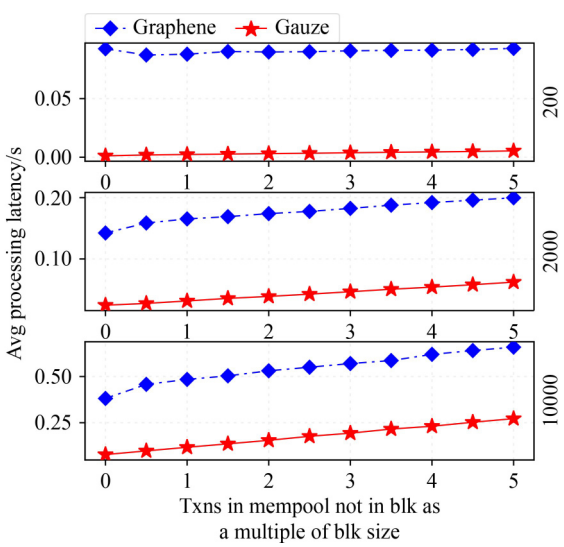

Fig. 7 Average processing latency (s) of Gauze VS. Graphene. Each facet is a block size: (200, 2000, and 10000 transactions)

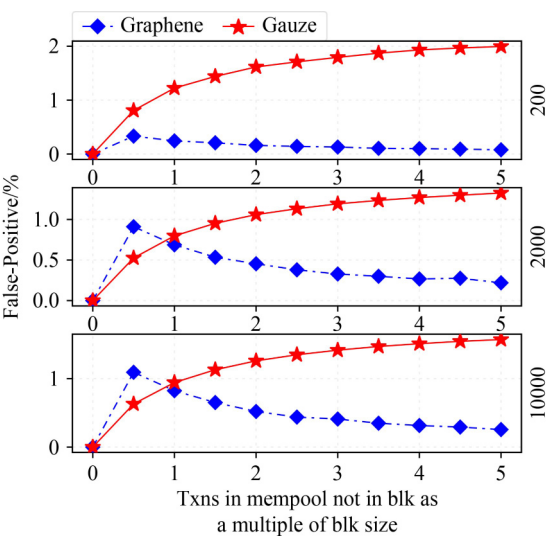

Fig. 9 Average false positive rate of Gauze VS. Graphene. Each facet is a block size: (200, 2000, and 10000 transactions)

0. When multiple increases, the false positive rate will appear. We can see that the changing trend of the false positive rate for Graphene is decreasing because of the use of both BF and IBLT data structures. The false positive rate of Gauze increases as the mempool increases and is higher than that of Graphene. This is because we set the fingerprint length to only 8 bits. If we want to reduce the false positive rate, we need to increase the fingerprint length.

Figures 10−12 show the average processing latency, the space cost and the false positive rate respectively, when the fingerprint length f is set as 16. The average processing latency is almost the same as the above experiment when $f = 8$, but the space overhead is doubled. The false positive rate decreases to almost zero. These experiments show that if we want higher accuracy, we can increase the length of the fingerprint. Although this will increase some space overhead, it hardly affects the processing efficiency.

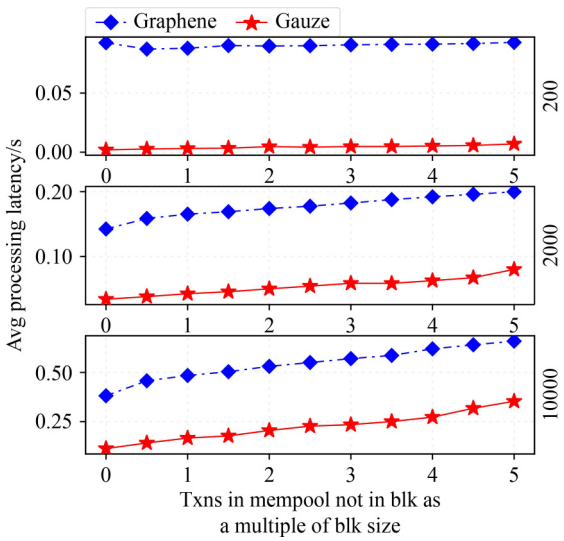

Fig. 10 Average processing latency (s) of Gauze VS. Graphene ($f = 16$). Each facet is a block size: (200, 2000, and 10000 transactions)

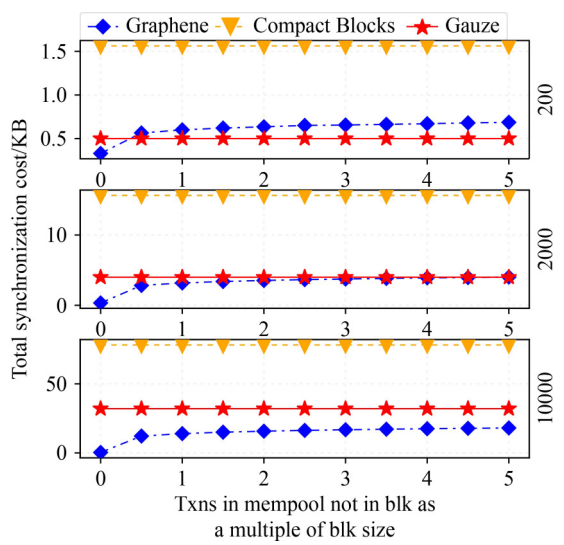

Fig. 11 Average space cost of Gauze VS. Compact Blocks and Graphene ($f = 16$). Each facet is a block size: (200, 2000, and 10000 transactions)

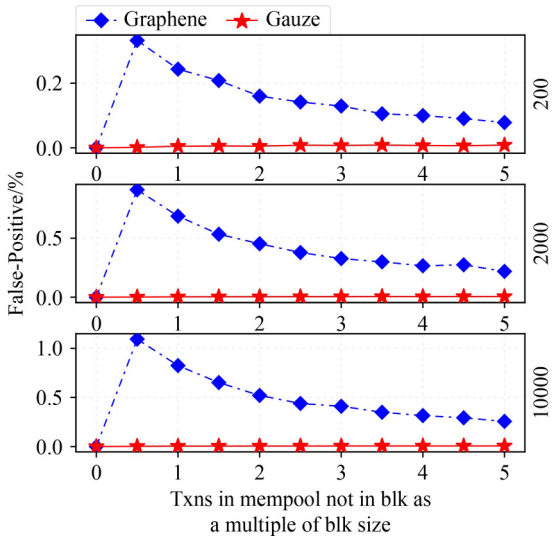

Fig. 12 Average false positive rate of Gauze VS. Graphene ($f = 16$). Each facet is a block size: (200, 2000, and 10000 transactions)

4.3 Evaluation of Gauze enhanced protocol

In these experiments, our main purpose is to evaluate our Gauze Enhanced protocol (Gauze's Protocol 2), which is designed for the second scenario where the mempool of the receiver does not possess all transactions in the block. The x-axis represents the fraction of transactions in the block contained in the receiver's mempool. Fraction is the ratio of transactions the block owned in the receiver's mempool. For example, at fraction 0.5 and block size 200, the mempool contains 300 transactions in total, which is $y = 100$ transactions in the block plus $m - y = 200$ additional transactions in the mempool (illustrated in Fig. 4). In this subsection, we focus on the second scenario, where the receiver does not possess the entire block ($m > n$) and evaluate $m = n$ as a particular case.

Local processing latency Figure 13 shows the cost of local processing latency in seconds of our Gauze enhanced protocol compared to graphene extended, as the fraction of the block

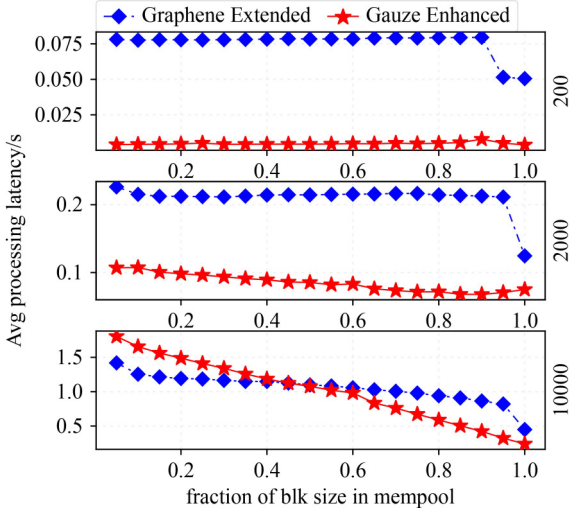

Fig. 13 Average processing latency (s) of Gauze Enhanced VS. Graphene Extended as the fraction of the block owned by the receiver increases

owned by the receiver increases. As the size of the mempool increases as a fraction of the block size, the processing latency of Graphene Extended is approximately 10× than that of Gauze Enhanced with a block size of 200. It is approximately 2× when the block size is 2,000. When the block size is 10,000, the Gauze Enhanced has poor performance because of the usage of the two CFs. It takes more time to query the large number of transactions in the mempool and insert them into the CFs, thus increasing the overall processing latency. In general, Gauze Enhanced is significantly better than Graphene Extended in terms of the processing performance for the same reasons as Protocol 1.

Total synchronization cost Figure 14 shows the average space cost in bytes of the Gauze Enhanced blocks compared to compact blocks and graphene extended blocks, as the fraction of the block owned by the receiver increases. When the block size is 200, the total synchronization cost of Gauze Enhanced is better than that of compact blocks and graphene extended. Moreover, when the block size is 2,000, the total synchronization cost of compact blocks is about 3× that of gauze enhanced, while Gauze Enhanced and Graphene Extended are roughly the same.

When the block size is 10,000, the total synchronization cost of Gauze Enhanced is better than Compact Blocks but slightly inferior to Graphene Extended. The reason for that is the number of buckets is set as the same as the block size. Gauze Enhanced has a more negligible overhead when the block is small. However, when the block size increases and two CFs are used in Protocol 2, the space overhead will increase in some cases, which is not as good as that of Graphene. The table length of the cuckoo filter needs to be limited to an exponential multiple of 2, which leads to the appearance of individual inflection points in Fig. 14. Equation (6) shows that cuckoo filters are better than Bloom filters in space cost. These experiments verify that Gauze Enhanced could significantly minimize the network bandwidth consumption and space cost required for the block synchronization process.

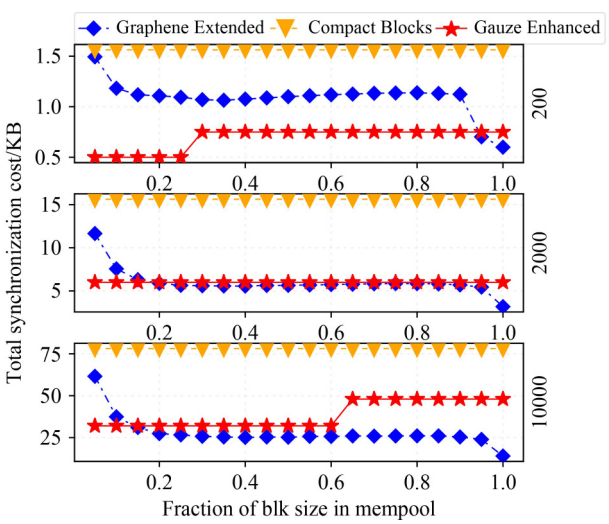

Fig. 14 Average space cost of Gauze Enhanced VS. Compact Blocks and Graphene Extended as the fraction of the block owned by the receiver increases

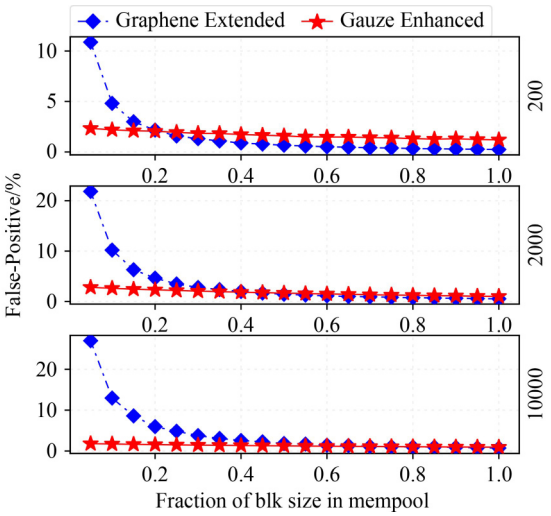

Fig. 15 Average false positive rate of Gauze Enhanced VS. Graphene Extended as the fraction of the block owned by the receiver increases

False positive rate Figure 15 shows the false positive rate of Gauze Enhanced compared to Graphene Extended, as the fraction of the block owned by the receiver increases. The results show that Gauze Enhanced and Graphene Extended show a relatively consistent trend in false positive rates, approximately below 2.0%. The false positive rates decrease as the fraction of the block owned by the receiver increases. We set the fingerprint length to only 8 bits for the above experiments. This means that if we want to reduce the false positive rate, we need to increase the fingerprint length.

4.4 Experimental conclusion

From the previous experiments, we achieve the following conclusions:

1) Gauze's local processing latency and processing performance thoroughly outperform that of Graphene in both scenarios. Gauze improves the efficiency of block synchronization during the block propagation process, allowing peers to process more transactions.

2) Gauze's total synchronization cost is less than that of Compact Blocks and Graphene in many scenarios. This is extremely important for blockchains because Gauze can save a large portion of the network bandwidth consumption and improve the network transmission efficiency. With fewer data being transferred during the block synchronization process, peers can pack more transactions into blocks, thus increasing the blockchain's throughput and scaling the blockchain system.

3) We set the fingerprint length f to 8 bits as the main purpose of this paper is to ensure better performance with a lower false positive rate. The false positive rates of Gauze are slightly higher than that of Graphene under certain scenarios and can be reduced by setting longer fingerprint length f.

5 Discussion

In this section, we further discuss several issues regarding the Gauze Protocol. Several uninvolved aspects of our Gauze Protocol warrant further discussion. We introduce them from

two design standpoints, which also suggest avenues for future work.

The decoding rate Since CF is a probabilistic data structure with the possibility of false positive rate, Gauze cannot decode the block 100% in all cases. The decoding rate of block synchronization reduces with the increase of the number of missed transactions in the receiver's mempool. In our experiments, 4.2, 4.3, Gauze's false positive rate is maintained at approximately 2%, so there will be a failure to decode due to these false positives. As given by Eq. (7), the decoding rate is relatively low with a high false positive rate. Reducing the false positive rate can increase the decoding rate, thus improving the success rate for block synchronization. We have currently used the method of increasing the fingerprint length to reduce false positive rate and improve the decoding rate. In future work, we will improve the decoding rate performance by adding new data structures such as a Marked Cuckoo Filter (MCF) or using an optimized algorithm design to find the best parameters(k, b, f).

Limitations Gauzes are employed for block synchronization. There are trade-offs among the block size, processing delay, number of transactions, transmission size, complexity (in terms of network round trips), and synchronization success rate in scenarios that the receiver missed transactions. By contrast, the current popular alternatives, such as Compact Blocks [18], have a predictable transmission size, fixed transmission complexity, use a trivial algorithm. Graphene [19] can also achieve a high success rate by using BFs and IBLTs, but the probabilistic data structure cannot achieve complete decoding in various scenarios. At present, Gauze cannot decode 100% of the data, but Gauze outperforms existing methods in terms of the average processing latency and the total synchronization space cost in different scenarios. Moreover, Gauze can achieve a high success rate without requiring extra roundtrips in complex scenarios. We do not claim that Gauze is the best solution for spreading blocks, nor do we claim to have a general blockchain expansion solution.

6 Conclusion

In this paper, We introduce Gauze, a novel solution to set reconciliation in block synchronization during block propagation in blockchain systems. The Cuckoo filter is utilized in Gauze to determine the subset of items jointly held by two parties from a more extensive set. By exchanging CF, our proposed Protocol 1 can determine the transactions contained in the block existing in the receiver's mempool for the scenario that the transactions in the block are a subset of the receiver's mempool. We also provide Protocol 2 for a more general situation where one party lacks some or all of the transactions in the block. By exchanging CFs, transactions that do not exist in the receiver's mempool can be transmitted to achieve block synchronization. The comprehensive experiments indicate that our approach requires less data transmission on the network and achieves higher processing performance than previous approaches.

Acknowledgements This work was supported in part by the National Natural Science Foundation of China (Grant No. 62032017).

References

1. Van Renesse R, Dumitriu D, Gough V, Thomas C. Efficient reconciliation and flow control for anti-entropy protocols. In: Proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware. 2008, 1−7
2. Kokoris-Kogias E, Jovanovic P, Gailly N, Khoffi I, Gasser L, Ford B. Enhancing bitcoin security and performance with strong consistency via collective signing. In: Proceedings of the 25th USENIX Conference on Security Symposium. 2016, 279–296
3. Gilad Y, Hemo R, Micali S, Vlachos G, Zeldovich N. Algorand: scaling byzantine agreements for cryptocurrencies. In: Proceedings of the 26th Symposium on Operating Systems Principles. 2017, 51–68
4. Buterin V, Griffith V. Casper the friendly finality gadget. 2017, arXiv preprint arXiv: 1710.09437
5. Ayinala K, Choi B Y, Song S. PiChu: accelerating block broadcasting in blockchain networks with pipelining and chunking. In: Proceedings of the 2020 IEEE International Conference on Blockchain (Blockchain). 2020, 221–228
6. Chawla N, Behrens H W, Tapp D, Boscovic D, Candan K S. Velocity: scalability improvements in block propagation through rateless erasure coding. In: Proceedings of the 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). 2019, 447–454
7. Zhang L, Wang T, Liew S C. Speeding up block propagation in bitcoin network: uncoded and coded designs. Computer Networks, 2022, 206: 108791
8. Imtiaz M A, Starobinski D, Trachtenberg A, Younis N. Churn in the bitcoin network: characterization and impact. In: Proceedings of the 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). 2019, 431–439
9. Croman K, Decker C, Eyal I, Gencer A E, Juels A, Kosba A, Miller A, Saxena P, Shi E, Sirer E G, Song D, Wattenhofer R. On scaling decentralized blockchains. In: Proceedings of the 2016 International Conference on Financial Cryptography and Data Security. 2016, 106–125
10. Luo L, Guo D, Li W, Zhang T, Xie J, Zhou X. Compound graph based hybrid data center topologies. Frontiers of Computer Science, 2015, 9(6): 860–1874
11. Decker C, Wattenhofer R. Information propagation in the bitcoin network. In: Proceedings of the IEEE P2P 2013 Proceedings. 2013, 1–10
12. Eppstein D, Goodrich M T, Uyeda F, Varghese G. What's the difference?: Efficient set reconciliation without prior context ACM SIGCOMM Computer Communication Review, 2011, 41(4): 218–229
13. Tschipper P. Buip010: xtreme thinblocks. See Bitco.in/forum/threads/buip010-passed-xtreme-thinblocks774 website, 2016
14. Bloom B H. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, 1970, 13(7): 422–426
15. Corallo M. Bip152: Compact block relay, See Github/bitcoin/bips/blob/master/bip-0152.mediawiki website, 2016
16. Ozisik A P, Andresen G, Levine B N, Tapp D, Bissias G, Katkuri S. Graphene: efficient interactive set reconciliation applied to blockchain propagation. In: Proceedings of the ACM Special Interest Group on Data Communication. 2019, 303–317
17. Goodrich M T, Mitzenmacher M. Invertible bloom lookup tables. In: Proceedings of the 2011 49th Annual Allerton Conference on

Communication, Control, and Computing (Allerton). 2011, 792–799
18. Fan B, Andersen D G, Kaminsky M, Mitzenmacher M D. Cuckoo filter: practically better than bloom. In: Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies. 2014, 75–88
19. Toomim J. Benefits of ltor in block entropy encoding, or: ISPs hate him! Learn how to make your block 75% Xthinner with this one weird trick. See Jtoomim.medium/benefits-of-ltor-in-block-entropy-encoding-or-8d5b77cc2ab0 website, 2018
20. Naumenko G, Maxwell G, Wuille P, Fedorova A, Beschastnikh I. Erlay: efficient transaction relay for bitcoin. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. 2019, 817–831
21. Shafeeq S, Zeadally S, Alam M, Khan A. Curbing address reuse in the iota distributed ledger: a cuckoo-filter-based approach. IEEE Transactions on Engineering Management, 2020, 67(4): 1244–1255
22. Fan B, Andersen D G, Kaminsky M. MemC3: compact and concurrent MemCache with dumber caching and smarter hashing. In: Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation. 2013, 371–384
23. Rottenstreich O. Sketches for blockchains. In: Proceedings of the 13th International Conference on COMmunication Systems & NETworkS (COMSNETS). 2021, 254–262
24. Guo D, Li M. Set reconciliation via counting bloom filters. IEEE Transactions on Knowledge and Data Engineering, 2013, 25(10): 2367–2380
25. Chen D, Konrad C, Yi K, Yu W, Zhang Q. Robust set reconciliation. In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. 2014: 135–146
26. Luo L, Guo D, Zhao Y, Rottenstreich O, Ma R T B, Luo X. MCFsyn: a multi-party set reconciliation protocol with the marked cuckoo filter. IEEE Transactions on Parallel and Distributed Systems, 2021, 32(11): 2705–2718
27. Ruan M, Titcheu T, Zhai E, Li Z, Liu Y, Jinlong E, Cui Y, Xu H. On the synchronization bottleneck of openstack swift-like cloud storage systems. IEEE Transactions on Parallel and Distributed Systems, 2018, 29(9): 2059–2074
28. Eppstein D. Cuckoo filter: simplification and analysis. In: Proceedings of the 15th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2016). 2016, 8

Xiaoqiang Ding is currently pursuing his MS degree from the College of Intelligence and Computing, Tianjin University, China. His research interests include data structure, distributed networking systems, and blockchain.

Liushun Zhao received his BE degree in communications engineering from Harbin Engineering University, China in 2018. From 2018, He continues to pursue his PhD degree in the School of Computer Science and Technology, Xidian University, China. His research interest focuses on security supervision in the operation process of blockchain.

Lailong Luo received the BS, MS, and PhD degrees from the College of Systems Engineering, National University of Defense Technology, China in 2013, 2015 and 2019, respectively. He is currently a lecturer in the College of Systems Engineering, National University of Defense Technology, China. His research interests include data structure and distributed networking systems.

Junjie Xie received the BE degree in computer science and technology from the Beijing Institute of Technology, China in 2013. He received the MS and PhD degrees in management science and engineering from the National University of Defense Technology, China in 2015 and 2020, respectively. He is currently an engineer with the Institute of Systems Engineering, AMS, PLA, Beijing, China. His research interests include distributed systems, software-defined networking, and mobile edge computing.

Deke Guo received the BS degree in industry engineering from the Beihang University, China in 2001, and the PhD degree in management science and engineering from the National University of Defense Technology, China, in 2008. He is currently a professor with the College of Systems Engineering, National University of Defense Technology, China. His research interests include distributed systems, software-defined networking, data center networking, wireless and mobile systems, and interconnection networks. He is a senior member of the IEEE and a member of the ACM.

Jinxi Li is currently pursuing his MS degree from the College of Intelligence and Computing, Tianjin University, China. His research interests include edge computing and network function virtualization.