AdaptChain: Adaptive Scaling Blockchain with Transaction Deduplication

Jie Xu, Qingyuan Xie, Sen Peng, Cong Wang, *Fellow, IEEE,* and Xiaohua Jia, *Fellow, IEEE*

Abstract—Although existing schemes improve blockchain throughput by allowing concurrent blocks to be appended to the blockchain, little attention has been devoted to adjusting blockchain throughput dynamically and deduplicating transactions between concurrent blocks. In this paper, we propose AdaptChain, an adaptive scaling blockchain with transaction deduplication. When the transaction demand of users in the network is high, the blockchain expands to meet the demand; when the transaction demand is low, the blockchain shrinks to save communication and storage costs. Our transaction deduplication mechanism ensures that no duplicate transactions are added to the blockchain, thereby improving bandwidth utilization and achieving higher effective throughput. Besides, we randomly split the mining power of the system to achieve mining power load balancing and resist attacks. We formally analyze the blockchain security and implement the proposed prototype on Amazon EC2. Experimental results show that AdaptChain achieves dynamic and higher effective blockchain throughput.

Index Terms—Distributed system, blockchain scalability, load balancing, transaction deduplication, consensus protocol.

1 Introduction

Blockchain consensus has attracted great attention in recent years [1–4]. The core of Bitcoin blockchain is Nakamoto consensus protocol [5]. Participants compete to solve Proof of Work (PoW) puzzles for the right to generate the next block, and honest nodes always follow the Longest Chain Rule (LCR). Although Nakamoto consensus protocol ensures blockchain security when adversary nodes control less than half of the total mining power in the system, it rejects concurrent blocks and thus limits the performance of the blockchain.

Extensive research has been devoted to improving blockchain scalability [6–8]. Some replace the single-chain structure with a Directed Acyclic Graph (DAG) [9–11], while others use parallel chains [12–14]. In DAG-based blockchains [9–11], a block can be pointed to by many concurrent child blocks, and a child block can also point to many parent blocks. DAG-based blockchains are vulnerable to double-spending attacks and liveness attacks [11]. In a parallel chain-based blockchain [12–14], there are multiple genesis blocks, and each genesis block is followed by a subchain. All subchains grow in parallel, and the mining difficulty must be related to the number of subchains to ensure the safety and liveness of the blockchain [13, 15].

While these schemes [9–14] increase the blockchain throughput by allowing concurrent blocks to be appended to the blockchain, they have two drawbacks. First, these schemes ignore the dynamic change of users' transaction demands and do not adjust the number of concurrent blocks accordingly. Blockchains do not shrink when the transaction demand is low, resulting in many near-empty blocks and incurring unnecessary communication and storage costs. Second, these schemes do not address the problem of duplicate transactions included in concurrent blocks. Since miners cannot be grouped in permissionless blockchains and miners prefer to pack transactions with higher transaction fees, blocks generated in parallel would include duplicate transactions. Duplicate transactions on the blockchain waste communication resources, increase the storage overhead of nodes and hinder blockchain scalability [16].

It is challenging for all nodes to dynamically scale up and down blockchain throughput according to the transaction demand in a distributed environment. First, it is difficult to synchronize to scale up and down consistently. Nodes have different local transaction pools due to network delay. If nodes decide to expand or shrink the blockchain based on transactions in their local transaction pools, it would lead to different actions. Second, existing blockchain structures, such as DAGs and parallel chains, do not support dynamically expanding or shrinking according to transaction demand. In a DAG-based blockchain, any number of child blocks can be appended to a block rather than the response to the surge of transaction demand. In a parallel chain-based blockchain, the number of subchains is fixed and cannot be changed according to dynamic transaction demand. Third, there is no trusted third party to allocate transactions unbiasedly. Nodes always tend to pack more profitable transactions. It requires a lot of communication to coordinate packing transactions to avoid duplication.

In this paper, we propose a PoW-based blockchain consensus protocol that adaptively adjusts the blockchain throughput according to the transaction demand, and no duplicate transactions are added to the blockchain. We propose the following schemes to address the challenges mentioned above: 1) We use the average block size of blocks that have reached consensus as an indicator of transaction demand. If the average block size of several consecutive blocks on the blockchain is close to the maximum block size, it indicates high transaction demand. If the average block size is small, it indicates low transaction demand. Thus, when the average block size is close to the maximum block

• *J.Xu, Q.Xie, S.Peng, C.Wang, and X.Jia is with the Department of Computer Science, City University of Hong Kong, Hong Kong. E-mail: jiexu49-c@my.cityu.edu.hk*

size, AdaptChain expands and the throughput increases. When the average block size is small, AdaptChain shrinks and the throughput decreases. 2) We design a blockchain structure that can expand or shrink exponentially. Multiple blocks can be appended to the blockchain in parallel in an epoch. When transaction demand is high, the blockchain expands, and the number of concurrent blocks in the next epoch is twice the number of blocks in the previous epoch. When transaction demand is low, the blockchain shrinks, and the number of concurrent blocks in the next epoch is half the number of blocks in the previous epoch. 3) We utilize the last bits of transaction hashes to deduplicate transactions in concurrent blocks. The last few bits of the PoW hashes of concurrent blocks are required to be different. Miners discard transactions whose last bits of the transaction hash differ from the last bits of the PoW hash before broadcasting the block, so there are no duplicate transactions among concurrent blocks.

In summary, the contributions of this paper are summarized as follows:

1) We propose AdaptChain, an adaptive scaling blockchain. When the transaction demand is high, the blockchain expands to meet transaction demand; when transaction demand is low, the blockchain shrinks to reduce communication and storage costs.
2) We design a mining power load balancing mechanism to split the mining power to subchains. Adversary nodes cannot concentrate their mining power to extend or attack a specific subchain so that AdaptChain can resist double-spending attacks, liveness attacks.
3) We design a transaction deduplication mechanism that can be used in permissionless environments. Blocks appended to the blockchain in parallel do not include duplicate transactions, which greatly increases effective blockchain throughput and bandwidth utilization.

2 Models and Problem Statement

2.1 System Model

AdaptChain is a PoW-based permissionless blockchain. Anyone can participate or quit the consensus protocol arbitrarily. All nodes are interconnected to a synchronous peer-to-peer network similar to other blockchain scaling protocols [11–13, 17]. Nodes communicate through a gossip protocol [18]. If an honest node broadcasts a block, other honest nodes will receive the block within a maximum network delay of δ [13].

New participating nodes establish connections with existing participating nodes and request the history of the blockchain from them to participate in the consensus protocol. Honest nodes validate received transactions and blocks before broadcasting them. They keep packing transactions and competing to generate new blocks. Each node i has a pair of public and private keys (pk_i, sk_i). Any node can use pk_i to validate the signature of the private key sk_i. Let $H(\cdot)$ denote the hash function and $H(m)$ denote the hash of the message m.

The hash function is modeled as a random oracle. Solving PoW puzzles can be modeled as querying a random oracle to find a parameter η that enables $H(h_{-1}, \eta, m)$ to have more than $log_2\frac{1}{p}$ leading zeros [13]. h_{-1} is the hash of the previous block on the blockchain, η is the solution to the PoW puzzle, and m is the content of the block. p is the mining difficulty, which means the probability of finding η to solve the puzzle by executing a hash operation is p [13].

Some participating nodes are controlled by probabilistic polynomial-time Byzantine adversary $\mathcal{A}$. The proportion of adversary nodes to all participating nodes is f, where $f < 0.5$. The rest are honest nodes, denoted by $\mathcal{H}$. The mining power of each node is the same. Adversary nodes can arbitrarily deviate from the protocol execution and employ various attack methods, including retransmission, out-of-order, delaying messages, etc. But adversary nodes cannot successfully modify messages without being discovered.

2.2 Problem Statement

We aim to design an adaptive scaling blockchain protocol that maximizes effective throughput while achieving safety and liveness. When the transaction demand is high, the blockchain throughput increases; when the transaction demand is low, the blockchain throughput decreases. There are no duplicate transactions on the blockchain. Safety and liveness are defined as follows:

- Safety: Once an honest node confirms a transaction on its local blockchain, any other honest node also confirms the transaction at the same position on its blockchain [19].
- Liveness: Any transaction from an honest node will eventually be confirmed by all honest nodes [19].

3 Design of Adaptive Scaling Blockchain Protocol with Deduplication of Transactions

3.1 Overview

AdaptChain proceeds in epochs. Initially, there is only one genesis block in epoch 0. Blocks with the same distance from the genesis block are in the same epoch.

The mining difficulty and the number of valid blocks in the next epoch are determined by the average block size in the previous epoch. If the average block size of blocks in the previous epoch is large, the mining difficulty decreases and AdaptChain expands. Each block in the previous epoch is appended by two new blocks in the next epoch so that the number of blocks in the next epoch is doubled. If the average block size of blocks in the previous epoch is small, the mining difficulty increases and AdaptChain shrinks. Each new block in the next epoch is appended to two blocks in the previous epoch so that the number of blocks in the next epoch is halved. Otherwise, the mining difficulty and the number of blocks in the next epoch remain unchanged. Each new block in the next epoch is appended to one of the blocks in the previous epoch.

For an epoch with 2^k concurrent blocks, the last k bits of PoW hash of the 2^k concurrent blocks are different from each other. A miner packs transactions from its local transaction pool, but only transactions whose last k bits of the transaction hash are the same as the last k bits of the PoW hash it solved remain. The other transactions of the block are dropped back into the pool. New blocks are appended

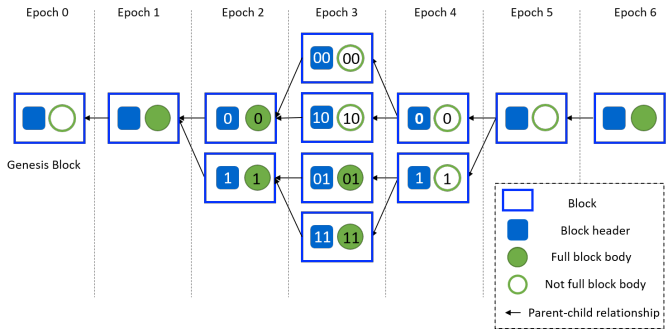


Fig. 1. Illustration of AdaptChain's adaptive scaling

to their parent blocks based on the last k bits of their PoW hashes.

In the following subsections, we first present the dynamic blockchain structure, including the mining power load balancing and mining difficulty adjustment that enables blockchain dynamic expansion and shrink. We then discuss the handling of duplicate and conflicting transactions. Finally, we present rules for blockchain consistency.

3.2 Dynamic Blockchain Structure

3.2.1 Concurrent Blocks in an Epoch

AdaptChain is similar to Nakamoto consensus protocol [5] employs PoW as the consensus mechanism. However, AdaptChain differs from the Nakamoto consensus protocol in that it allows 2^k valid blocks $B_1, B_2, ..., B_{2^k}$ to be appended to the blockchain in parallel in an epoch to achieve higher and dynamic throughput, where $k = 0, 1, 2, ...k_{max}$. Correspondingly, miners use the root R_{-1} of the Merkle tree composed of 2^k concurrent blocks in the previous epoch as one of the inputs to PoW puzzle $H(R_{-1}, \eta, m)$ instead of the hash of the previous block in Nakamoto consensus protocol. The 2^k concurrent blocks from the previous epoch are the *predecessors* of the new block.

Let BID denote $H(R_{-1}, \eta, m)$. Suppose there are 2^k concurrent blocks in an epoch. The last k bits of the BID of the 2^k concurrent blocks must be different. For example, as shown in Figure 1, when $k = 2$, the last 2 bits of the BID of 4 concurrent blocks must be 00, 10, 01, and 11, respectively. Miners cannot mine blocks in the next epoch until they have received all the 2^k concurrent blocks in the current epoch. If a node receives multiple blocks with the same last k bits of BID in this epoch, the first received block is appended to the blockchain. After a node receives all 2^k blocks, it can determine whether the blockchain expands, shrinks, or grows in parallel according to the average block size.

Threshold for blockchain expansion. It should be noted that the size of a single block in an epoch cannot indicate the transaction demand of users because adversary nodes may maliciously generate empty blocks to reduce system efficiency. In AdaptChain, the transaction demand in an epoch is jointly determined by all valid blocks in previous w epochs on the blockchain. The average block size of the blocks from epoch $e - w$ to epoch $e - 1$ is used to indicate the transaction demand in epoch e. To make the blockchain throughput meet the transaction demand timely, $w = 1$ in this paper. This means miners determine the number of valid blocks in epoch e based on the blocks in epoch $e - 1$.

Specifically, let $\|B\|$ denote the maximum block size, e.g. $\|B\| = 1$ MB for Bitcoin block. If the average size of blocks in an epoch is greater than $\alpha_u \|B\|$, the state of the epoch is full. α_u is the Threshold for the full epoch state, and $0.5 < \alpha_u \leq 1$. Once the epoch is full, the blockchain expansion is triggered, and the number of blocks in the next epoch will be doubled.

Threshold for blockchain shrink. If the average size of blocks in an epoch is less than $\alpha_d \|B\|$, the state of the epoch is empty. α_d is the Threshold for the empty epoch state, and $0 \leq \alpha_d \leq 0.5$. Once the epoch is empty, the blockchain shrink is triggered, and the number of blocks in the next epoch will be halved.

Two cases for blockchain parallel growth. 1) The average block size in an epoch is less than $\alpha_u \|B\|$ but greater than $\alpha_d \|B\|$. 2) The number of concurrent blocks in the previous epoch has reached $2^{k_{max}}$, and the epoch is still full. $2^{k_{max}}$ is the maximum number of concurrent blocks, which is limited by network bandwidth. Due to the limitations of network bandwidth and network delay, blockchains cannot expand indefinitely. After reaching $2^{k_{max}}$ concurrent blocks, the blockchain grows in parallel if the epoch is still full. The number of blocks in the next epoch is the same as the number of blocks in the current epoch.

3.2.2 Mining Power Load Balancing Mechanism

Mining power load balancing is crucial to prevent liveness attacks in DAG-based blockchains. Adversary nodes can strategically reserve or release their blocks to make blockchain loses liveness [11]. Additionally, the unbalanced growth of the blockchain hinders the adaptive throughput adjustment of the blockchain. To address this, AdaptChain introduces a mining power load balancing mechanism.

At a high level, AdaptChain balances the load of the blockchain by randomly appending new blocks based on the new block's randomly generated BID. Only predecessors in the previous epoch whose BID shares the same last bits as the new block's BID are considered parent blocks. Blocks are only appended after their parent blocks and not all predecessors, as shown in Figure 1.

To explain this mechanism more concretely, suppose there are 2^k blocks in epoch e. When the blockchain expands, there will be 2^{k+1} blocks in epoch $e+1$. Each block in epoch e is the parent of two subsequent blocks in epoch $e+1$ whose BID shares the same last k bits of its BID. When the blockchain shrinks, there will be 2^{k-1} blocks in epoch $e+1$. Each block in epoch $e+1$ has two parent blocks in epoch e. The parent blocks are the two blocks in the previous epoch whose BID shares the same last $k-1$ bits of the subsequent block's BID. When the blockchain grows in parallel, there will be 2^k blocks in epoch $e+1$. Each block in epoch $e+1$ has one parent block in epoch e. The parent block is the block in the previous epoch whose BID shares the same last k bits of the subsequent block's BID.

For example, as shown in Figure 1, the subsequent blocks in epoch 3 whose last 2 bits of the BID are 00 and 10 are appended to the block in epoch 2 whose last bit of BID is 0, because the last bit of their BID is 0. The block in epoch 2 whose last bit of BID is 0 is the parent of blocks in epoch

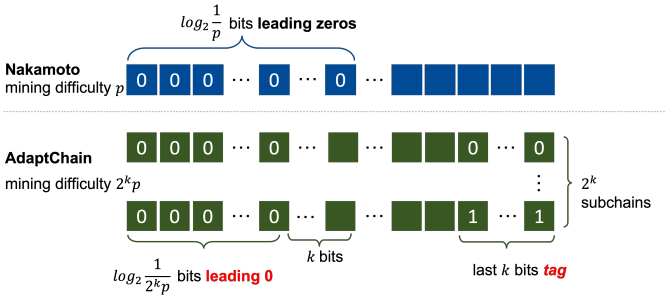


Fig. 2. PoW hash values for Nakamoto consensus protocol and AdaptChain with 2^k concurrent subchains

3 whose last 2 bits of the BID are 00 and 10. The block in epoch 4 whose last bit of BID is 1 is appended to blocks in epoch 3 whose last 2 bits of the BID are 01 and 11 because the last bit of their BID is 1. Both blocks in epoch 3 whose last 2 bits of the BID are 01 and 11 are the parent of block in epoch 4 whose last bit of BID is 1.

The generation of a new block's BID is random. Miners cannot predetermine the BID of the new block before solving the PoW puzzle, so they cannot predetermine parent blocks of a new block. Let *subchain* denote a sequence of blocks starting from the genesis block and following the parent-child relationship one block per epoch to block at the end of the blockchain. The mechanism ensures that any new block is randomly appended to a subchain, resulting in a balanced allocation of mining power on all subchains. Adversary nodes cannot concentrate their mining power to extend one of the subchains, thus ensuring the security of the blockchain. Because BIDs are randomly generated during the mining process and included in new blocks, the mining power load balancing mechanism does not introduce additional communication, computation, and storage overhead.

3.2.3 Adjustment of Mining Difficulty

Although the number of concurrent blocks changes dynamically, the total mining power of the whole system remains more or less constant. In order to keep the same level of difficulty for mining a block relative to the total available mining power mining this block, the mining difficulty needs to be adjusted each time when the blockchain expands or shrinks. When the blockchain expands, the mining difficulty should decrease. When the blockchain shrinks, the mining difficulty should increase.

As shown in Figure 2, in Nakamoto consensus protocol, the mining difficulty p refers to the probability of finding a PoW hash with $log_2\frac{1}{p}$ leading zeros by executing a hash operation is p. Given block size $\|B\|$, network delay δ and mining difficulty p, the block generation interval I can be determined so that Nakamoto consensus protocol satisfies safety and liveness [13, 19, 20]. Thus, a mining difficulty can be mapped to a block generation interval.

In AdaptChain, we dynamically adjust the blockchain throughput by adjusting the mining difficulty to meet dynamic transaction demands. It is expected that during a block generation interval I, 2^k blocks are appended to the blockchain in parallel instead of only one block. When there are 2^k concurrent blocks in an epoch, the last k bits of PoW hashes are used as *tag* for appending new blocks[1] as shown in Figure 2, and the mining difficulty is $2^k p$. The difficulty of generating a block to extend a given subchain by executing the hash operation once is the same as Nakamoto consensus protocol because $log_2\frac{1}{2^k p}+k = log_2\frac{1}{p}$, so that the block interval is I. Since subchains are extended in parallel, 2^k blocks are appended to the blockchain in parallel during a block generation interval instead of only one block. Therefore, we can dynamically adjust the blockchain throughput by adjusting the mining difficulty to meet dynamic transaction demands.

Before mining, miners determine the mining difficulty and the number of blocks in the epoch based on the number and state of the previous epoch. We explain the dynamic adjustment of mining difficulty in the following three cases.

Adjustment of mining difficulty at blockchain expansion. If there are 2^k blocks in the previous epoch and the previous epoch is full, the blockchain expands. There would be 2^{k+1} concurrent blocks/subchains in the next epoch, and the mining difficulty is $2^{k+1}p$. Because a new block is randomly appended to one of the 2^{k+1} concurrent subchains, the probability that a successfully mined block to be appended to a specific subchain on the blockchain is $\frac{1}{2^{k+1}}$. The probability of successfully mining a block and appending it to a specific subchain by executing a hash operation is shown as Eq.(1).

$$2^{k+1}p \cdot \frac{1}{2^{k+1}} = p \tag{1}$$

Therefore, the probability of executing a hash operation to generate a block for a specific subchain is exactly p, the same as in Nakamoto consensus. The block generation interval of this epoch is I, while the blockchain throughput is 2^{k+1} times that of the Nakamoto consensus protocol.

Adjustment of mining difficulty at blockchain shrink. If there are 2^k blocks in the previous epoch and the previous epoch is empty, the blockchain shrinks. There would be 2^{k-1} concurrent blocks/subchains in the next epoch, and the mining difficulty is $2^{k-1}p$. Because a new block is randomly appended to one of the 2^{k-1} concurrent subchains, the probability that a successfully mined block to be appended to a specific subchain on the blockchain is $\frac{1}{2^{k-1}}$. The probability of successfully mining a block and appending it to a specific subchain by executing a hash operation is shown as Eq.(2).

$$2^{k-1}p \cdot \frac{1}{2^{k-1}} = p \tag{2}$$

Therefore, the block generation interval of this epoch is I, and the blockchain throughput is 2^{k-1} times that of the Nakamoto consensus protocol.

Adjustment of mining difficulty at parallel growth. When the blockchain grows in parallel with 2^k concurrent blocks/subchains, the mining difficulty is $2^k p$. There would be 2^k concurrent blocks/subchains in the next epoch, and the mining difficulty remains $2^k p$. Each new block is randomly appended to one of the 2^k concurrent subchains. The probability that a successfully mined block is appended to a specific subchain on the blockchain is $\frac{1}{2^k}$. When the blockchain grows in parallel, the probability of successfully

1. In fact, any k bits of a PoW hash other than leading zeros can be used as a tag.

mining a block and appending it to a specific subchain by executing a hash operation is shown as Eq.(3).

$$2^k p \cdot \frac{1}{2^k} = p \tag{3}$$

Therefore, the block generation interval of this epoch is I, and the blockchain throughput is 2^k times that of the Nakamoto consensus protocol.

3.3 Processing of Transactions

3.3.1 Transaction Deduplication Mechanism

Some blockchain scaling schemes [9–14] allow for blocks to be generated and appended to the blockchain in parallel. However, such parallel block generation can result in duplicate transactions among concurrent blocks, as miners tend to pack transactions with higher transaction fees. It is a waste of communication and storage resources to add duplicate transactions on the blockchain, which can ultimately harm the effective throughput of the blockchain.

To address this issue, we propose a transaction deduplication mechanism to eliminate the inclusion of duplicate transactions on the blockchain and enhance effective throughput. The deduplication of transactions is a complex challenge. Existing DAG-based blockchains [9–11] can only ensure that there are no duplicate transactions in blocks of different epochs by excluding transactions in blocks that have been received. Sharding-based blockchains [21, 22] assign transactions to shards whose identities match those of the transactions to avoid duplicate transaction processing. But there are no determined shard/subchain identities in AdaptChain, and even the number of subchains is dynamic. Moreover, there is no trusted third party to allocate transactions impartially. It is not realistic for miners to coordinate packing transactions to achieve transaction deduplication. Miners are selfish and distrust each other. They will not disclose the transactions they have received honestly. Distributed communication for transaction deduplication costs lots of network bandwidth, reducing consensus efficiency.

Our proposed transaction deduplication mechanism uses the last k bits of the Transaction Identity (TID) to allocate transactions within an epoch containing 2^k blocks. We eliminate duplicate transactions across concurrent blocks by requiring each block only to contain the transactions whose last k bits of the TID are the same as the last k bits of its BID. The transaction deduplication mechanism includes two steps: 1) generating a raw Merkle tree before solving the puzzle; 2) discarding some subtrees of the raw Merkle tree after solving the puzzle.

First, miners each construct a Merkle tree. As shown in Figure 3, the raw Merkle tree consists of 2^k subtrees. Each subtree consists of transactions with the same last k bits of TID, and these transactions must not have been added to the blockchain in previous epochs. Transactions in different subtrees have different last k bits of TID. The maximum size of a subtree is $\|N_t\|$. $\|N_t\|$ denotes the size of the Merkle tree when a block with only one Merkle tree reaches the maximum block size.

Second, miners prune raw Merkle trees. After a miner solves the puzzle, it knows the last k bits of the BID. $2^k - 1$ subtrees where the last k bits of the TID is different from the last k bits of the BID are discarded. The rest of the raw Merkle tree remains as the block body. Thus, in AdaptChain, the block body is not a complete Merkle tree. For example, if a miner solves the puzzle with the last 2 bit of BID 00, the gray dotted parts in Figure 3 are discarded, and only the green solid-line parts remain in the block body. Some subtrees are discarded to ensure that the block size does not exceed the maximum block size, even if the raw Merkle tree expands 2^k times. The path and root of the Merkle tree are preserved to allow others to verify the PoW puzzle.

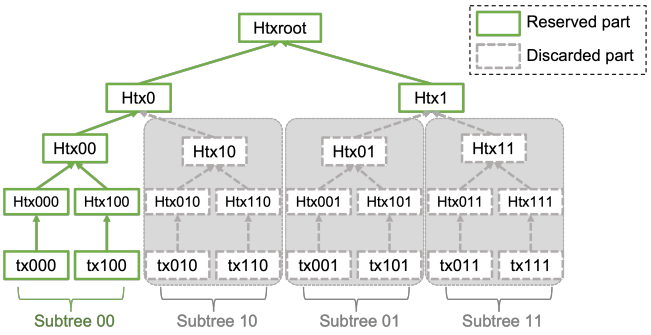


Fig. 3. Deduplication of transactions

Therefore, only transactions whose last k bits of the TID are the same as the last k bits of the block's BID remain in the block body. Because the last k bits of the BID of concurrent blocks are different, the transactions included in the concurrent blocks are different. There are no duplicate transactions among blocks in the same epoch.

3.3.2 Handling of Conflicting Transactions

Although the transaction deduplication mechanism ensures no duplicate transactions on the blockchain, there may still be conflicting transactions. For example, an adversary node generates multiple double-spending transactions using the same Unspent Transaction Output (UTXO). We handle conflicting transactions by first optimistically allowing conflicting transactions to be added to the blockchain and then determining a unique order of transactions. Only the first transaction in the order of conflicting transactions is valid. Other transactions are invalid even if they have been appended to the blockchain.

The order of transactions in different epochs follows the order of the epochs. The smaller the epoch, the higher the priority. If two transactions are in the same epoch but in different blocks, the order of the transactions follows the order of the blocks. The order of blocks in the same epoch with 2^k blocks is determined by the last k bits of their BID. The smaller the value of each bit from back to front, the higher the priority. For example, when there are 4 blocks in an epoch ($k = 2$), the last 2 bits of the BID of the 4 blocks must be 00, 10, 01, and 11, respectively. First, 00 and 10 have higher priority than 01 and 11 because the last bit $0 <$ 1. Then, according to the penultimate bit, 00 has a higher priority than 10, and 01 has a higher priority than 11. Thus, the order of the four blocks is 00, 10, 01, and 11. Besides, if conflicting transactions are included in the same block, the block is invalid and discarded.

3.4 Rules for Blockchain Consistency

Due to network delay, when a puzzle is solved, and a new valid block is generated, other nodes may continue to solve the puzzle and generate other new blocks in parallel. Even if honest nodes strictly follow the consensus protocol, nodes that receive blocks in different orders have different local blockchains. A node's local blockchain is its current blockchain view. To ensure the consistency of the blockchain, only one blockchain view that represents the majority of the mining power is authoritative. Honest nodes always extend the authoritative blockchain.

Nakamoto consensus protocol determines the representative blockchain view by the Longest Chain Rule (LCR) [5]. The longest blockchain represents the most mining power invested and is the authoritative blockchain. Honest miners always extend the longest chain. However, Sompolinsky *et al.* argue that LCR cannot represent the majority of the mining power and is vulnerable to double-spending attacks when the block generation rate is high [23]. They propose a secure resolution for inconsistent blockchain views called Greedy Heaviest Observer SubTree (GHOST) [23], which is commonly used in various DAG-based blockchains [11, 24]. GHOST requires that honest nodes always select the heaviest subtree rooted at the fork as the authoritative blockchain.

We adopt a variant of GHOST to determine the authoritative blockchain with two differences. First, the mining power is determined by all blocks after the fork because it may not be a subtree structure after the fork. Second, the mining power is not simply represented by the number of blocks but by the sum of the weighted mining power of all blocks after the fork. The mining power of a block with mining difficulty $2^k p$ is

$$w_k = \frac{1}{2^k p}. \tag{4}$$

When there are different blockchain views after a fork, honest nodes always select the blockchain view with more mining power after the fork as the authoritative blockchain and continue to extend it. If the two blockchain views after the fork have the same weight, nodes choose the first received one as the authoritative blockchain.

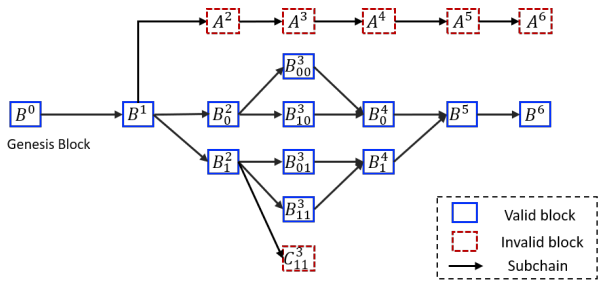


Fig. 4. AdaptChain's rules for blockchain consistency

For example, in Figure 4, there is a fork at block B^1. One is B^1, A^2, A^3, A^4, A^5, A^6, and its mining power is

$$6 \times \frac{1}{p} = \frac{6}{p}.$$

The second is B^1, B^2_0, B^2_1, B^3_{00}, B^3_{10}, B^3_{01}, B^3_{11}, C^3_{11}, B^4_0, B^4_1, B^5, B^6, and its mining power is

$$3 \times \frac{1}{p} + 4 \times \frac{1}{2p} + 5 \times \frac{1}{4p} = \frac{25}{4p}.$$

The mining power of C^3_{11} needs to be considered as it is contributed to (B^2_0, B^2_1). Because $\frac{6}{p} < \frac{25}{4p}$, honest nodes select the second as the authoritative blockchain and continue to extend it.

Similarly, there are two blockchain views after (B^2_0, B^2_1). One is B^2_0, B^2_1, B^3_{00}, B^3_{10}, B^3_{01}, B^3_{11}, B^4_0, B^4_1, B^5, B^6, and its mining power is

$$4 \times \frac{1}{2p} + 4 \times \frac{1}{4p} + 2 \times \frac{1}{p} = \frac{5}{p}.$$

The second is B^2_0, B^2_1, B^3_{00}, B^3_{10}, B^3_{01}, C^3_{11}, and its mining power is

$$2 \times \frac{1}{2p} + 4 \times \frac{1}{4p} = \frac{2}{p}.$$

Because $\frac{5}{p} > \frac{2}{p}$, honest nodes select the first blockchain view as the authoritative blockchain and continue to extend it.

Miners also store blocks of other blockchain views for a few epochs in case the blockchain view with less mining power evolves into the blockchain with the most mining power in the future. If the heaviest blockchain view changes, miners will switch to the heaviest blockchain view and continue mining. Blocks that are not in the heaviest blockchain view can be discarded if blocks of the same epoch have already been confirmed.

If an adversary node attempts to modify a block on the blockchain, it must provide more proof of work that has been done for the target block and all subsequent blocks after the target block. The difficulty of reversing a block increases exponentially as subsequent blocks are appended. The more epochs that extend after a block, the higher the confidence to confirm a block. Let T denote the threshold for block confirmation. If more than T epochs follow a block, the probability of the block being modified is negligible.

4 Implementation of the Adaptive Scaling Blockchain

4.1 Initializations

After the genesis block is released, each participating node in the blockchain network maintains its local transaction pool and local blockchain. When a new node joins the network, it establishes peer-to-peer connections with existing nodes and updates its local blockchain by requesting the latest blockchain from its peers. All participating nodes listen for new blocks in the network and validate them. Valid blocks are appended to their local blockchain and broadcast to their peer nodes.

The local transaction pool caches transactions that nodes have received and validated but have not yet appended to the blockchain. When a node receives a transaction, it first determines whether it has processed the transaction based on the TID. If the node has already processed the transaction, the transaction will be discarded. If not processed, the node first validates the transaction. Valid transactions are added to its local transaction pool and broadcast to its

peer nodes. Otherwise, invalid transactions are discarded. Transactions already on the blockchain will be removed from the transaction pool. Nodes delete transactions based on transaction fees or other criteria when their transaction pools reach the maximum size.

Each participating node updates its local blockchain through a mining thread and a receiving thread. The mining thread keeps solving PoW puzzles. If a puzzle is successfully solved, the node generates a new block and broadcasts it. The receiving thread processes messages received from other peer nodes to synchronize the blockchain. If a node receives a new block, it validates the block and appends the valid block to its local blockchain.

4.2 Mining Thread

Nodes generate new blocks with the algorithm 1. Initially, the basic mining difficulty is p_0 when there is only one valid block in an epoch.

Each node adaptively adjusts the mining difficulty of the current epoch according to the blocks in the previous epoch before mining. If the blocks in the previous epoch are full and the number of concurrent blocks in the previous epoch has not reached $2^{k_{max}}$, the number of concurrent blocks is doubled, and the mining difficulty of the next epoch decreases (line 2-3). If the blocks in the previous epoch are empty and the number of concurrent blocks in the previous epoch is greater than 1, the number of blocks halves, and the mining difficulty of the next epoch increases (line 4-5). Otherwise, the blockchain grows in parallel, and the mining difficulty is the same as in the previous epoch (line 7).

Algorithm 1: Adaptive Block Generation

Input: Basic mining difficulty: p_0,
BID in the previous epoch: $\mathcal{B} = \{B_1, B_2, ..., B_{2^k}\}$,
Number of blocks in the previous epoch: 2^k,
Txs for the current epoch: $\mathcal{T} = \{tx_1, tx_2, ..., tx_n\}$
Output: New Blocks

1 $\bar{B} =$ AVERAGESIZE($\mathcal{B}$)
2 **if** $\bar{B} > \alpha_u \|B\|$ && $2^k < 2^{k_{max}}$ **then**
3 $\quad k \leftarrow k+1, \quad p \leftarrow 2^k p_0$ // Blockchain expands
4 **else if** $\bar{B} < \alpha_d \|B\|$ && $2^k > 1$ **then**
5 $\quad k \leftarrow k-1, \quad p \leftarrow 2^k p_0$ // Blockchain shrinks
6 **else**
7 $\quad p \leftarrow 2^k p_0$ // Blockchain grows in parallel
8 **end**
9 $B.predecessors \leftarrow$ PREDECESSORSHASH($\mathcal{B}$)
10 $B.root \leftarrow$ MERKLEROOT($\mathcal{B}$)
11 $B.tx \leftarrow$ MERKLETREE($\mathcal{T}$)// Raw Merkle tree
12 MINING (B, p)

Procedure 2 describes the mining process. The mining difficulty p corresponds to the hash of the PoW puzzle with $log_2\frac{1}{p}$ leading zeros. The prepared new block components $Btemp$ and $nonce$ are the inputs to the hash function HASH(). INCREMENTNONCE() continuously changes $nonce$ to compute new hash BID until BID has more than $log_2\frac{1}{p}$ leading zeros if the stop mining request SHUTDOWNREQUESTED() is not received (line 5-7). If a BID has no less than $log_2\frac{1}{p}$ leading zeros, the last k bits of the BID are used as the tag to determine the parent blocks of the new block. DEDUPTX() deletes the transactions in the raw Merkle tree whose last k bits are different from the tag. BLOCKASSEMBLER() assembles $nonce$ and the block after removing duplicate transactions to generate a new block $newB$ (line 13). PROCESSBLOCK() further processes $newB$ and adds it to the local blockchain.

Procedure 2: Mining(B, p)

1 $d \leftarrow log_2\frac{1}{p}$
2 $nonce \leftarrow$ NEWNONCE()
3 $Btemp \leftarrow$ CREATSEMIBLOCK($B, ChainParams$)
4 $BID \leftarrow$ HASH($Btemp, nonce$)
5 **while** *BID has less than d leading zeros* && !SHUTDOWNREQUESTED() **do**
6 $\quad nonce \leftarrow$ INCREMENTNONCE($nonce$)
7 $\quad BID \leftarrow$ HASH($Btemp, nonce$)
8 **end**
9 **if** *BID has no less than d leading zeros* **then**
10 $\quad B.bid \leftarrow BID$
11 $\quad B.tag \leftarrow$ last k bits of BID
12 $\quad B.detx \leftarrow$ DEDUPTX($B.tx, tag$)
$\quad$ // Deduplicate txs in $B.tx$ based on tag
13 $\quad newB \leftarrow$ BLOCKASSEMBLER($B, Btemp, nonce$)
14 $\quad$ PROCESSBLOCK($newB$)
15 **end**

4.3 Receiving Thread

Nodes process the received block as shown in Procedure 3. A received block can be valid only if all its previous blocks of the block are valid. Nodes request previous blocks that are not available locally from their peer nodes with REQUESTPEERS() and process them with PROCESSBLOCK() (line 1-7).

For each unprocessed block, a node first determines the mining difficulty of the block according to the state $state$ and the number of blocks n in the previous epoch (line 8-10). Then, CHECKPOW() validates the proof of work of the new block, and CHECKTX() validates the transactions of the block. If any invalid, block processing aborts and returns False. Otherwise, the block is appended to the local blockchain.

Each valid block has two pointers to its parent blocks. If the blockchain shrinks, the two parent blocks are blocks in the previous epoch with the same last bits of BID as the processing block's tag. The two pointers of the block point to the two parent blocks, respectively (line 12-17). If the blockchain expands or grows in parallel, each block has only one pointer to its unique parent block (line 19-22). The other pointer is a null pointer (line 23). Finally, the node updates its local block set (line 25) after appending the block to its local blockchain.

4.4 Rules for Blockchain Consistency

Every time a new block is generated, or a new block is received, the node's local blockchain graph changes. The node determines the mining power of the blockchain views

Procedure 3: ProcessBlock(B)

```
1  foreach B' ∈ B.predecessors do
2      if B' ∉ G then
3          RequestPeers(B')
4          if !ProcessBlock(B') then
5              return False
6          end
7      end
8  end
9  n ← CountUpper(B.predecessors)
10 size ← SizeUpper(B.predecessors)/n
11 state ← StateUpper(size)
12 if CheckPoW(B, state, n)&&CheckTx(B) then
13     if state=shrink then
14         l ← log2 n − 1
15         for B1, B2 ∈ B.predecessors do
16             if the last l bit of B1.bid = the last l bit of
                B2.bid = B.tag then
17                 B.parent1 ← B1
18                 B.parent2 ← B2
19             end
20         end
21     else
           // Blockchain expands or grows in parallel
22         l ← log2 n
23         for each B1 ∈ B.predecessors do
24             if the last l bit of B1.bid= B.tag then
25                 B.parent1 ← B1
26             end
27         end
28         B.parent2 ← NULL
29     end
30     G ← G ∪ B // Update the local blockchain
       return G
31 else
32     return False
33 end
```

Algorithm 4: Rules for Blockchain Consistency (G)

Input: Received blockchain graph G
Output: Authoritative blockchain θ

```
1  S ← ∅ // Collect all valid sibling groups to S
2  foreach B ∈ G do
3      s ← SiblingGroup(B)
4      S.Append(s)
5  end
6  foreach s ∈ S do
7      W(s) ← 0
8  end
9  foreach B ∈ G do
10     foreach s ∈ S do
11         foreach v ∈ SubviewsBeginWith(s) do
12             if B ∈ v then
13                 W(s) ← W(s) + MiningPower(B)
14                 break
15             end
16         end
17     end
18 end
   // Find the authoritative blockchain
19 s ← Siblings(B^0) // B^0 stands for genesis block
20 θ ← {s}
21 while ChildSiblingGroups(s) ≠ ∅ do
22     Update s ← arg max_{s' ∈ ChildSiblingGroups(s)} W(s')
23     θ.Append(s)
24 end
```

Function: SubviewsBeginWith(s)

```
1  V ← ∅
2  if ChildSiblingGroups(s) = ∅ then
3      v ← ConstructSubview(s)
4      V ← V.Append(v)
5      return V
6  else
7      foreach c ∈ ChildSiblingGroups(s) do
8          foreach v ∈ SubviewsBeginWith(c) do
9              V.Append(v.Append(s))
10         end
11     end
12 end
```

included in the updated blockchain graph through Algorithm 4 and further determines the authoritative blockchain with the most mining power.

Blocks with the same predecessor are *siblings*. The 2^k siblings whose mining difficulty is $\frac{1}{2^k p}$ and the last k bits of their BID are different form a *sibling group*, where $k = 0, 1, 2, ...k_{max}$. Algorithm 4 first collects all sibling groups in the blockchain graph to S (line 2-5). Let *subview* of a sibling group s denote a sequence of sibling groups beginning with s and following the predecessors-children relationship to the end of the blockchain. Function SubviewsBeginWith(s) is used to find all subviews that begin with s. If the blocks in s are the predecessors of blocks in another sibling group s', then s' is the child sibling group of s. Function ChildSiblingGroups(s) is used to find child sibling groups of s. According to the rules for blockchain consistency, the mining power of a block B, which is MiningPower(B), is contributed to the weight of all subviews $W(s)$ that contain it (line 7-14). Similar to GHOST [23], line 15-20 follow a path from the genesis block and select the sibling group leading to the heaviest subview at each fork. All selected sibling groups construct the authoritative blockchain θ.

5 Security Analysis

5.1 Safety

Safety means all honest users have a consistent blockchain view for confirmed blocks. Similar to Nakamoto protocol, the blockchain in our scheme is $T-consistent$ [19]. Honest nodes agree on the blocks on the blockchain except for unconfirmed blocks in the last T epochs of the blockchain.

Our proposed scheme applies a variant of GHOST rule whereby the sum of the weighted mining power of blocks determines the authoritative blockchain. Regardless of the

Function: CHILDSIBLINGGROUPS(s)

```
C ← ∅
foreach s' ∈ S do
    foreach B' ∈ s' do
        foreach B ∈ s do
            if B ∈ B'.predecessors then
                C.APPEND(s')
            end
        end
    end
end
```

number of blocks in an epoch, the sum of the weighted mining power of all blocks in an epoch is $\frac{1}{p}$. If these blocks are considered as a whole, our authoritative blockchain selection rule is equivalent to the original GHOST rule [23]. Therefore, the security properties of the GHOST rule and the proofs are also applicable to our scheme.

Proposition 1 (Convergence of History)**.** *For any block B, it is either abandoned or adopted by all nodes at the earliest time ψ_B. Then, $\Pr(\psi_B < \infty) = 1$ and $E(\psi_B) < \infty$. In other words, each block must be eventually either fully adopted or fully abandoned.*

Proof. The underlying network is synchronous, which means that if a node broadcasts a block at time t_1, all nodes in the network will receive the block at or before $t_1 + \delta$. Assume that at time t block B is neither abandoned or adopted by all nodes. Let ε_t denote the event that the next new block is generated between $t + \delta$ and $t + 2\delta$, and no other block is generated until time $t + 3\delta$. All nodes learn all existing blocks between time t and $t + \delta$. The two blockchain views that nodes are extending simultaneously must be two subviews of the same mining power after a fork on the blockchain. The next new block is generated so that ties for the two blockchain views are broken, and all nodes learn it after another δ time. Therefore, nodes switch to one of the subviews. Honest nodes generate blocks with constant mining power, and block generation time follows an exponential distribution. According to the properties of the exponential distribution, $\Pr(\varepsilon_t)$ is independent of t and lower bounded by a positive number [23]. The expected time for the first ε_t to occur for the first time is finite. Thus, t is upper bounded. $\Pr(\psi_B < \infty) = 1$ and $E(\psi_B) < \infty$. □

We next demonstrate that our blockchain consistency rule is resistant to 50% attacks. Even with a small block generation interval, the probability of an accepted block changing to be abandoned after a sufficiently long time is negligible.

Similar to previous works [5, 23, 25], we model the block mining process as a Poisson process. Specifically, when the mining difficulty is $2^k p$, the number of blocks that can be generated by executing n hash operation is $M_k(n)$. $M_k(n)$ is a Poisson process with intensity $2^k p$. According to the properties of the Poisson process, the expectation and variance are

$$\mathrm{E}[M_k(n)] = 2^k pn, \mathrm{Var}[M_k(n)] = 2^k pn. \tag{5}$$

Theorem 1 (Weight Expectation and Variance)**.** *Given the number of hash operations n, the weight of mining power is a random variable, denoted $W(n)$. The expectation of $W(n)$ is independent of the mining difficulty and is identically equal to n. That is, $E[W(n)] = n$, and the variance is $\mathrm{Var}[W(n)] \leq \frac{n}{p}$.*

Proof. Let n_k $(0 \leq k \leq k_{\max})$ denote the number of hash operations n used to generate blocks with mining difficulty $2^k p$. Then, the total number of hash operations can be represented as:

$$n = n_0 + n_1 + \cdots + n_{k_{\max}}. \tag{6}$$

Analogously, the weight of mining power $W(n)$ is

$$W(n) = W_0(n_0) + W_1(n_1) + \cdots + W_{k_{\max}}(n_{k_{\max}}). \tag{7}$$

According to rules for blockchain consistency, all the generated blocks contribute to the weight of the blockchain, and the weight of a block with mining difficulty $2^k p$ is

$$w_k = \frac{1}{2^k p}. \tag{8}$$

Thus, the total weight of generated blocks is:

$$W_k(n_k) = M_k(n_k) w_k = \frac{M_k(n_k)}{2^k p}. \tag{9}$$

Combining Eq. (5), the expectation and variance of $W_k(n_k)$ are

$$\begin{aligned} \mathrm{E}[W_k(n_k)] &= \frac{\mathrm{E}[M_k(n_k)]}{2^k p} = \frac{2^k p n_k}{2^k p} = n_k, \\ \mathrm{Var}[W_k(n_k)] &= \frac{\mathrm{Var}[M_k(n_k)]}{(2^k p)^2} = \frac{n_k}{2^k p}. \end{aligned} \tag{10}$$

Therefore, the expectation of the total weight is

$$\begin{aligned} \mathrm{E}[W(n)] &= E\left[\sum_{0 \leq k \leq k_{\max}} W_k(n_k)\right] \\ &= \sum_{0 \leq k \leq k_{\max}} \mathrm{E}[W_k(n_k)] = \sum_{0 \leq k \leq k_{\max}} n_k = n. \end{aligned} \tag{11}$$

Since $\{W_0(n_0), W_1(n_1), ..., W_k(n_k)\}$ are independent of each other, the total weight variance

$$\begin{aligned} \mathrm{Var}[W(n)] &= \mathrm{Var}\left[\sum_{0 \leq k \leq k_{\max}} W_k(n_k)\right] \\ &= \sum_{0 \leq k \leq k_{\max}} \mathrm{Var}[W_k(n_k)] = \sum_{0 \leq k \leq k_{\max}} \frac{n_k}{2^k p} \\ &\leq \sum_{0 \leq k \leq k_{\max}} \frac{n_k}{p} = \frac{n}{p}. \end{aligned} \tag{12}$$

□

Let F denote the event that the 50% attack is successful. This means that the blockchain generated by adversary nodes has more mining power than the blockchain generated by honest nodes, which is $F = \{W_a - W_h > 0\}$.

Lemma 1. *Let $\Delta = \mathrm{E}[W_h] - \mathrm{E}[W_a]$. Then,*

$$F \subseteq \{\mathrm{E}[W_h] - W_h > \frac{\Delta}{2}\} \cup \{W_a - \mathrm{E}[W_a] > \frac{\Delta}{2}\}.$$

Proof.

For all $\omega \notin \{\mathrm{E}[W_h]-W_h > \frac{\Delta}{2}\}\cup\{W_a-\mathrm{E}[W_a] > \frac{\Delta}{2}\}$, then

$$\omega \in \{\mathrm{E}[W_h] - W_h \leq \frac{\Delta}{2}\} \cap \{W_a - \mathrm{E}[W_a] \leq \frac{\Delta}{2}\}. \quad (13)$$

According to Assertion(13),

$$\omega \in \{\mathrm{E}[W_h] - W_h + W_a - \mathrm{E}[W_a] \leq \frac{\Delta}{2} + \frac{\Delta}{2} = \Delta\}. \quad (14)$$

Because $\Delta = \mathrm{E}[W_h] - \mathrm{E}[W_a]$, Assertion(14) can be transformed into

$$\omega \in \{W_a - W_h \leq 0\}. \quad (15)$$

Thus, $\omega \notin F = \{W_a - W_h > 0\}$, which means

$$F \subseteq \{\mathrm{E}[W_h] - W_h > \frac{\Delta}{2}\} \cup \{W_a - \mathrm{E}[W_a] > \frac{\Delta}{2}\}. \quad (16)$$

□

Theorem 2 (Resilience to 50% Attacks)**.** *Suppose the mining power of honest nodes is v_h, and the mining power of adversary nodes is $q \cdot v_h$. If $q \in [0, 1)$, $\forall \epsilon > 0, \exists t_0 > 0$, s.t. $\forall t > t_0$, $\Pr[F] < \epsilon$.*

Proof. According to Lemma 1,

$$\begin{aligned}\Pr[F] &\leq \Pr[\{\mathrm{E}[W_h] - W_h > \frac{\Delta}{2}\} \cup \{W_a - \mathrm{E}[W_a] > \frac{\Delta}{2}\}] \\ &\leq \Pr[\mathrm{E}[W_h] - W_h > \frac{\Delta}{2}] + \Pr[W_a - \mathrm{E}[W_a] > \frac{\Delta}{2}] \\ &\leq \Pr[|\mathrm{E}[W_h] - W_h| > \frac{\Delta}{2}] + \Pr[|W_a - \mathrm{E}[W_a]| > \frac{\Delta}{2}].\end{aligned} \quad (17)$$

According to Chebyshev's inequality,

$$\begin{aligned}\Pr[F] &\leq \frac{\mathrm{Var}[W_h]}{(\Delta/2)^2} + \frac{\mathrm{Var}[W_a]}{(\Delta/2)^2} \\ &= \frac{4(\mathrm{Var}[W_h] + \mathrm{Var}[W_a])}{\Delta^2}.\end{aligned} \quad (18)$$

According to Theorem 1,

$$\begin{aligned}\Delta &= \mathrm{E}[W_h] - \mathrm{E}[W_a] \\ &= n_h - n_a = v_h t - q v_h t = (1-q) v_h t.\end{aligned} \quad (19)$$

$$\mathrm{Var}(W_h) + \mathrm{Var}(W_a) = \frac{n_h + n_a}{p} = \frac{(1+q)v_h t}{p}. \quad (20)$$

Combining Eq.(19) and Eq.(20), Eq.(18) can be transformed into:

$$\mathrm{P}(F) \leq \frac{4(1+q)v_h t}{p[(1-q)v_h t]^2} = \frac{4(1+q)}{p(1-q)^2 v_h}\frac{1}{t}. \quad (21)$$

$\forall \epsilon > 0$, let

$$t_0 = \lceil \frac{4(1+q)}{p(1-q)^2}\frac{1}{v_h \epsilon} \rceil.$$

For all $t > t_0$, we can have

$$\mathrm{P}(F) < f(t_0) \leq \frac{4(1+q)}{p(1-q)^2 v_h} \Big/ \frac{4(1+q)}{p(1-q)^2}\frac{1}{v_h \epsilon} = \epsilon. \quad (22)$$

Therefore, the probability of successful 50% attacks can be negligible. □

5.2 Liveness

Liveness states that any transaction accepted by an honest node will eventually be added to the local blockchain of all honest nodes after enough time. The original GHOST [23] is vulnerable to liveness attacks when the block generation rate is high [11]. The liveness attack refers to adversary nodes preventing honest nodes from reaching a consensus on a blockchain by maintaining the balance of the two blockchain views. Because of network delay, when the block generation rate is high, honest nodes may not receive in-transit blocks in time. There are two sets of honest nodes, each extending a blockchain view. Adversary nodes simultaneously extend two competing blockchains and strategically hold/release blocks with a small amount of mining power to keep the two blockchains of equal weight. Thus, the two groups of honest nodes always consider their blockchain authoritative. Adversary nodes can prevent two blockchains from merging so that liveness requirements are not met [11].

In contrast, the mining power balancing mechanism in AdaptChain can resist liveness attacks by splitting the adversary nodes' mining power, even if the block generation interval is short. The mining power balancing mechanism randomly appends new blocks to one of the subchains. Because adversary nodes cannot determine the parent block of its generated block before solving the puzzle, the blocks generated by adversary nodes may be invalid blocks. Adversary nodes cannot successfully balance different blockchains with a small amount of mining power. Therefore, adversary nodes cannot balance the two blockchain views, and our proposed blockchain protocol is resistant to liveness attacks.

6 Experimental Evaluation

We implemented AdaptChain in C++ and deployed it on the Amazon EC2 platform using 20 Amazon EC2 m4.2xlarge instances, distributed across 14 different cities worldwide. Each instance was equipped with 2 vCPUs, 32 GB of memory, and a 1000 Mbps network interface. To evaluate the performance of large-scale nodes, we simulated 50 participating nodes on each instance, resulting in a total of 1000 participating nodes in our experimental evaluation.

Each node was configured to connect randomly to eight other nodes. Thus, the underlying blockchain network is a P2P network with eight peer nodes per node. We use TCP protocol push/pull to transmit messages between nodes. When a node generates a new block, it first broadcasts the new block to its eight peer nodes. The peer nodes then forwarded the block to their peer nodes after validation.

6.1 Choosing Block Size

The throughput and latency of a blockchain consensus protocol vary under different parameters. Block size is a crucial parameter affecting blockchain throughput and Block Propagation Delay (BPD). We first adjust the block size in order to identify an optimal size for achieving higher protocol performance. Figure 5 presents the throughput and BPD of the blockchain for varying block sizes, ranging from 20 KB to 1024 KB, in a single-chain structure with a 128-second block generation interval.

As depicted in Figure 5, when the block size is less than 600 KB, the blockchain throughput increases with block

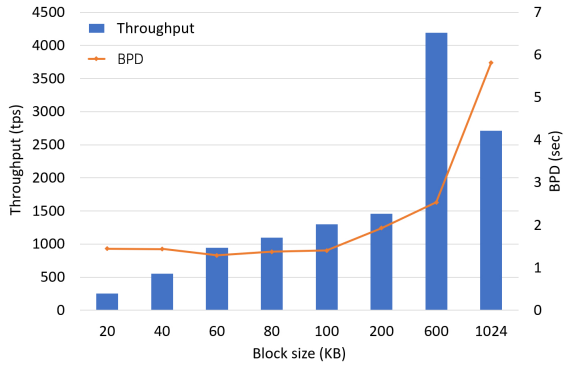


Fig. 5. The impact of block size on blockchain throughput and BPD

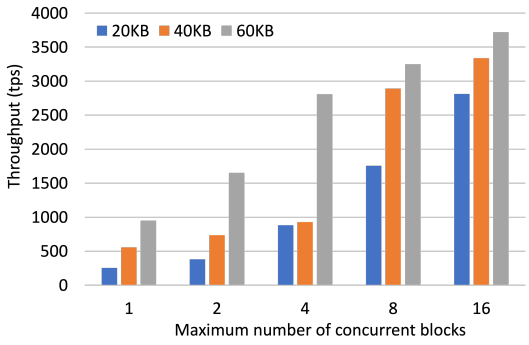


Fig. 6. The impact of the number of concurrent blocks and block size on blockchain throughput

size. However, when the block size reaches 1024 KB, the throughput decreases due to the longer time required to broadcast the block across the network. When the block size is 1024 KB, and the block generation interval is 128 seconds, network congestion may occur. Consequently, generated blocks cannot be broadcast to all participating nodes in time, decreasing blockchain throughput. Figure 5 also reveals that the difference in BPD is minimal when the block size is between 20KB and 100KB. When the block size exceeds 200 KB, the BPD significantly increases with the block size. Moreover, Awe observed that larger blocks lead to more temporary forks, as miners may start working on different views of the blockchain before receiving the latest block. This results in wasted computing resources and reduced overall throughput.

Blockchains that only add one valid block to the chain per epoch generally opt for larger blocks to achieve higher throughput, such as the 1 MB to 2 MB blocks used in Algorand [26]. AdaptChain and OHIE [13] scale the blockchain with concurrent blocks. Based on the preliminary experiment regarding block size, we can conclude that a large block significantly negatively impacts BPD. Consensus protocols with multiple blocks propagated in parallel favor multiple small blocks over one large block to increase throughput [13]. Considering the results of Figure 5 and various factors, we primarily evaluate blockchain throughput and BPD for block sizes between 20 KB and 60 KB in the following experiments.

6.2 Dynamic Scalability

We conduct three sets of experiments with block sizes of 20 KB, 40 KB, and 60 KB to assess the impact of blockchain expansion and contraction. We simulate mining difficulty adjustment by controlling the block generation interval for each node, similar to OHIE [13]. Initially, the expected value of the block generation interval is set to 128 seconds for a single-chain structure. The thresholds for blockchain expansion, α_u, and shrink, α_d, are set to 0.85 and 0.15, respectively. Each time the blockchain expands, the expected value of the block generation interval is halved. Each time the blockchain shrinks, the expected value of the block generation interval doubles.

Each experiment begins with a single genesis block and a 128-second block generation interval. The transaction pool is large enough to ensure that miners constantly generate full blocks, causing the blockchain to expand continuously. In each experiment, we set the maximum number of concurrent blocks to 1, 2, 4, 8, and 16 and measure the throughput as the blockchain gradually scales to the maximum number of concurrent blocks and stabilizes. When the maximum number of concurrent blocks is 1, Adaptcain degenerates to Nakamoto consensus protocol. This data set is used for comparison. The average transaction size is 500 Bytes, and the network bandwidth is 20 Mbps.

As shown in Figure 6, blockchain throughput increases with blockchain expansion when the block size is fixed. Notably, the increase is exponential at a 20 KB block size. Under Nakamoto consensus protocol with a block size of 20 KB, the blockchain throughput is approximately 251 transactions per second (tps). When the blockchain expands to 16 concurrent blocks, the throughput reaches up to 2810 tps, roughly 11 times that of the single-chain structure. The reasons for not achieving a 16-fold increase include: 1) the network becoming congested and network delays increasing as the number of concurrent blocks increases, and 2) throughput being the average throughput of the blockchain as the number of concurrent blocks expands from 1 to 16. Therefore, AdaptChain can achieve adaptive throughput scaling at smaller block sizes compared to the Bitcoin blockchain with the Nakamoto consensus protocol.

Compared to OHIE [13] using the same experimental platform at 20 Mbps bandwidth and 20 KB block size, AdaptChain has similar blockchain throughput as OHIE. However, OHIE is a solution with a fixed number of parallel blockchains, and its throughput cannot be adjusted after the system is initialized. Our scheme can achieve a higher throughput ceiling and reduce the storage overhead of nodes when transaction demand is low.

6.3 Bandwidth

To evaluate the impact of bandwidth on blockchain scaling, we conduct experiments where we fixed the block size at 20 KB and the maximum number of concurrent blocks at 16. To control the bandwidth of the socket, we use the socket() function to adjust the buffer size, which indirectly regulates the amount of data that could be sent or received at one time. We then measure the blockchain throughput and BPD by varying the available bandwidth of each node to 8 Mbps, 12 Mbps, 16 Mbps, and 20 Mbps, respectively.

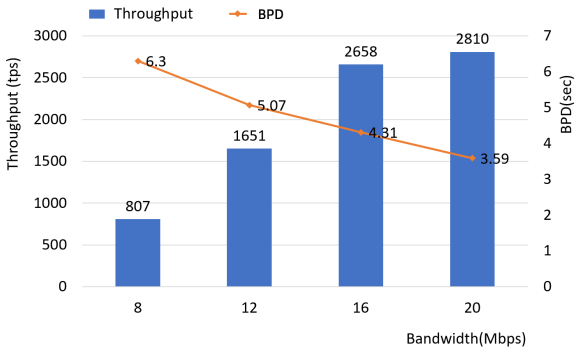


Fig. 7. The impact of the network bandwidth on blockchain throughput and BPD

Figure 7 shows that as bandwidth increases, the blockchain throughput increases while BPD decreases. Increased bandwidth enhances block transmission and reduces latency, resulting in higher throughput.

6.4 Confirmation Latency

Blocks in the last T epochs on the blockchain are unconfirmed blocks, and blocks before the last T epochs on the blockchain are confirmed blocks. As T increases, the probability of a confirmed block being modified or deleted decreases exponentially. Nodes can choose different T values according to their confidence in the block confirmation. T affects only block confirmation latency and has no impact on blockchain throughput and BPD. Confirmation latency is the time from when a block is appended to the blockchain to when it becomes a confirmed block. Figure 8 shows the average confirmation latency for different T values when the block size is 20 KB, and the bandwidth is 20 Mbps.

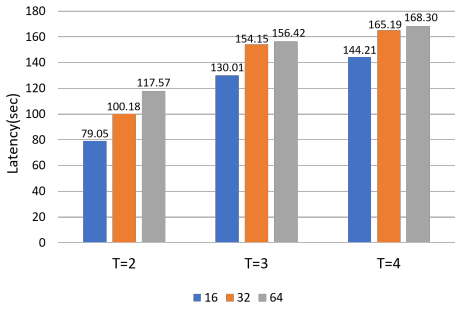


Fig. 8. Confirmation latency

The results in Figure 8 indicate that as T increases, confirmation latency increases since a larger T requires waiting for more epochs to append blocks to the blockchain. If the value of T is fixed, the more concurrent blocks, the longer the confirmation latency. This is because more concurrent blocks take longer to be added to the blockchain. When $T = 4$ and there are 64 concurrent blocks, the confirmation latency is about 168.3 seconds. This result is roughly the same as OHIE [13], but AdaptChain can achieve adaptive throughput compared with OHIE, and there are no duplicate transactions on AdaptChain.

7 Related Work

DAG-based blockchains. Some studies [9–11, 23, 24, 27] have explored using a Directed Acyclic Graph (DAG) structure to improve blockchain throughput by allowing for an arbitrary number of concurrent blocks. For example, Sompolinsky *et al.* [23] proposed GHOST to replace LCR as the blockchain conflict resolution solution, which accumulates the mining power of the blocks on the subtree of the fork when nodes choose the main chain. Spectre [9], and Phantom [10] attempt to improve throughput by adopting a DAG structure but suffer from global transaction serialization [12] and liveness attacks [11], respectively. Conflux [11] uses the GHOST rule to determine a pivot chain in the DAG for achieving global transaction serialization and has two strategies for generating blocks to deal with liveness attacks while increasing throughput. However, Conflux is difficult to shrink, resulting in the waste of storage and communication costs when the transaction demand is low. Our prior work, Occam [27], dynamically adjusts throughput based on actual transaction demands. Compared to Occam, AdaptChian proposes new rules based on GHOST to ensure blockchain consistency, enhance blockchain security at high throughput, and provide detailed theoretical analysis. AdaptChain also has three outstanding advantages compared to other DAG-based blockchains: 1) The transaction deduplication mechanism ensures there are no duplicate transactions on AdaptChain, maximizing the effective throughput of the blockchain. 2) AdaptChain shrinks when the transaction demand is low to reduce the storage and communication overhead of nodes. 3) AdaptChain ensures load balancing of mining power, randomly splitting the mining power of adversary nodes and preventing targeted attacks on specific subchains. These features make AdaptChain resistant to both 50% and liveness attacks, making it a promising solution for secure and high-throughput blockchain applications.

Parallel chains. Parallel chains improve blockchain throughput by simultaneously extending several chains in parallel, each with an independent genesis block. OHIE [13] is a representative solution of parallel blockchains, which is coupled by multiple parallel instances of Nakamoto Consensus. Each new block contains the hash of all the blocks at the end of parallel chains. The last bits of the hash of the new block determine which chain the block is appended to, preventing adversary nodes from focusing on attacking a subchain. Block confirmation in OHIE takes a long time since block confirmations need to wait for all parallel chains to have new blocks before moving forward. OHIE achieves a global order of blocks across all parallel chains by using chains' IDs and reference relationships between blocks. Compared with OHIE, AdaptChain can achieve adaptive throughput according to transaction demand.

Parallel chains in Chainweb [14] are tightly coupled by reference to each other, but it does not have a random block allocation mechanism, making subchains vulnerable. Kiffer *et al.* [20] pointed out that Chainweb cannot improve blockchain performance as effectively as it claims. They designed a variant of Chainweb, Cliquechain. Cliquechain only achieves the same throughput as Nakamoto consensus protocol, while Cliquechain provides provable consistency.

AdaptChain randomly splits the mining power of adversary nodes so that they cannot concentrate on extending or attacking a specific subchain. As proved in security analysis, as long as the mining power of adversary nodes is less than half of the total mining power of the system, AdaptChain can resist 50% attack and liveness attack. Different from Cliquechain, AdaptChain's difficulty adjustment mechanism reduces the mining difficulty when the blockchain expands, thereby increasing the blockchain throughput. Prism [12] decouples the function of blocks into transaction blocks for transaction packing, proposal blocks for leader election, and voter blocks for block confirmation. Prism accelerates block confirmation by introducing multiple parallel voting chains that extend simultaneously. Compared to AdaptChain, there are a large number of duplicate transactions in Prism's concurrent transaction blocks, which wastes network bandwidth.

The number of chains in these schemes [12–14, 20] is fixed at the time of initialization, which causes unnecessary waste of resources when the transaction demand in the network is low. AdaptChain offers a more flexible solution compared to previous schemes. Unlike the fixed number of parallel chains in these schemes, AdaptChain's number of chains can adjust according to transaction demand, reducing the storage and communication overhead of nodes. Blockchain throughput and network bandwidth consumption are low when transaction demand is low. When transaction demand is high, the blockchain throughput increases up to the maximum allowed by the network bandwidth.

8 Conclusion

While most of the existing research makes efforts to improve the maximum throughput of blockchains, we alternatively focus on the dynamics and effectiveness of blockchain throughput. This paper presents AdaptChain, an adaptive scaling blockchain protocol with transaction deduplication. The blockchain throughput changes dynamically according to transaction demand. When the transaction demand is high, the mining difficulty decreases, and the blockchain expands. When the transaction demand is low, the mining difficulty increases, and the blockchain shrinks. Our proposed transaction deduplication mechanism ensures that blocks appended to the blockchain in parallel do not contain duplicate transactions, increasing the effective throughput of the blockchain and reducing communication and storage costs. The security analysis confirmed the safety and liveness of AdaptChain. The experimental results show that the blockchain throughput changes dynamically.

References

[1] M. Saad, Z. Qin, K. Ren, D. Nyang, and D. Mohaisen, "e-PoS: Making proof-of-stake decentralized and fair," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 8, pp. 1961–1973, 2021.

[2] C. Xu, K. Wang, P. Li, S. Guo, J. Luo, B. Ye, and M. Guo, "Making big data open in edges: A resource-efficient blockchain-based approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 4, pp. 870 – 882, 2019.

[3] H. Huang, Z. Yue, X. Peng, L. He, W. Chen, H.-N. Dai, Z. Zheng, and S. Guo, "Elastic resource allocation against imbalanced transaction assignments in sharding-based permissioned blockchains," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 10, pp. 2372–2385, 2022.

[4] X. Fu, H. Wang, and P. Shi, "Votes-as-a-proof (vaap): Permissioned blockchain consensus protocol made simple," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4964–4973, 2022.

[5] S. Nakamoto. (2008) Bitcoin: A peer-to-peer electronic cash system. [Online]. Available: http://www.bitcoin.org/bitcoin.pdf

[6] W. Li, C. Feng, L. Zhang, H. Xu, B. Cao, and M. A. Imran, "A scalable multi-layer pbft consensus for blockchain," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 5, pp. 1146–1160, 2020.

[7] Y. Hassanzadeh-Nazarabadi, A. Küpçü, and Ö. Özkasap, "Lightchain: Scalable DHT-based blockchain," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 10, pp. 2582–2593, 2021.

[8] Z. Cai, J. Liang, W. Chen, Z. Hong, H.-N. Dai, J. Zhang, and Z. Zheng, "Benzene: Scaling blockchain with cooperation-based sharding," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 2, pp. 639–654, 2022.

[9] Y. Sompolinsky, Y. Lewenberg, and A. Zohar, "Spectre: A fast and scalable cryptocurrency protocol," Cryptology ePrint Archive, Report 2016/1159, 2016, https://ia.cr/2016/1159.

[10] X. Li, Y. Zheng, K. Xia, T. Sun, and J. Beyler, "Phantom: An efficient privacy protocol using zk-snarks based on smart contracts," Cryptology ePrint Archive, Report 2020/156, 2020, https://ia.cr/2020/156.

[11] C. Li, P. Li, D. Zhou, Z. Yang, M. Wu, G. Yang, W. Xu, F. Long, and A. C.-C. Yao, "A decentralized blockchain with high throughput and fast confirmation," in *Proceedings of the 2020 USENIX Annual Technical Conference*, 2020, pp. 515–528.

[12] V. Bagaria, S. Kannan, D. Tse, G. Fanti, and P. Viswanath, "Prism: Deconstructing the blockchain to approach physical limits," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2019, p. 585–602.

[13] H. Yu, I. Nikolić, R. Hou, and P. Saxena, "OHIE: Blockchain scaling made simple," in *Proceedings of the 2020 IEEE Symposium on Security and Privacy*. IEEE, 2020, pp. 90–105.

[14] W. Martino, M. Quaintance, and S. Popejoy, "Chainweb: A proof-of-work parallel-chain architecture for massive throughput," *Chainweb Whitepaper*, vol. 19, 2018.

[15] X. Wang, V. V. Muppirala, L. Yang, S. Kannan, and P. Viswanath, "Securing parallel-chain protocols under variable mining power," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 1700–1721.

[16] J. Xu, C. Wang, and X. Jia, "A survey of blockchain consensus protocols," *ACM Computing Surveys*, 2023.

[17] J. Garay, A. Kiayias, and N. Leonardos, "The bit-

coin backbone protocol: Analysis and applications," in *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 281–310.

[18] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, "Epidemic algorithms for replicated database maintenance," in *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, 1987, pp. 1–12.

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

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

[21] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, and E. Syta, "OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding," in *Proceedings of the 2018 IEEE Symposium on Security and Privacy*. IEEE, 2018, pp. 583–598.

[22] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 931–948.

[23] Y. Sompolinsky and A. Zohar, "Secure high-rate transaction processing in bitcoin," in *International Conference on Financial Cryptography and Data Security*. Springer, 2015, pp. 507–527.

[24] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.

[25] A. Dembo, S. Kannan, E. N. Tas, D. Tse, P. Viswanath, X. Wang, and O. Zeitouni, "Everything is a race and nakamoto always wins," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2020, pp. 859–878.

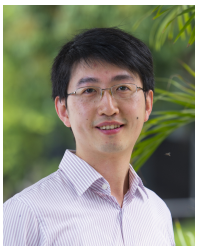
[26] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th symposium on operating systems principles*, 2017, pp. 51–68.

[27] J. Xu, Y. Cheng, C. Wang, and X. Jia, "Occam: A secure and adaptive scaling protocol for permissionless blockchain," in *Proceedings of the 41st IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE, July 2021, pp. 618–628.

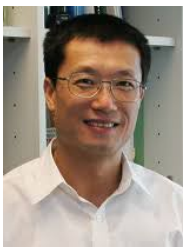
Jie Xu received the B.S. degree from the Department of Information Security, University of Science and Technology of China (USTC) in July 2017, and the M.S. degree from the Department of Electronic Engineering and Information Science (EEIS), USTC in 2020. She is currently pursuing the Ph.D. degree with the Department of Computer Science, City University of Hong Kong. Her research interests include network security and cryptography.

Qingyuan Xie received his B.E. degree in network engineering from the Anhui University, in 2017, the M.E. degree in computer science and technology from the Harbin Institute of Technology (Shenzhen), in 2020. He is currently pursuing the Ph.D. degree with the Department of Computer Science, City University of Hong Kong. His research interests include mobile edge computing, cloud security.

Sen Peng received his B.E. degree in computer science and technology from the SUSTech in 2020. He is currently pursuing the Ph.D. degree with the Department of Computer Science, City University of Hong Kong. His research interests include artificial intelligence security and cloud security.



Cong Wang (Fellow, IEEE) is currently a Professor with the Department of Computer Science, City University of Hong Kong. He is also with the City University of Hong Kong Shenzhen Research Institute. His research interests include data and network security, blockchain and decentralized applications, and privacy-enhancing technologies. He is a member of ACM. He has been one of the Founding Member of the Young Academy of Sciences of Hong Kong since 2017. He was conferred the RGC Research Fellow in 2021. He received the Outstanding Researcher Award (junior faculty) in 2019, the Outstanding Supervisor Award in 2017, and the President's Awards in 2016 and 2019, all from the City University of Hong Kong. He was a co-recipient of the Best Paper Award of IEEE ICDCS 2020, ICPADS 2018, MSN 2015, the Best Student Paper Award of IEEE ICDCS 2017, and the IEEE INFOCOM Test of Time Paper Award in 2020. His research has been supported by multiple government research fund agencies, including the National Natural Science Foundation of China, the Hong Kong Research Grants Council, and the Hong Kong Innovation and Technology Commission. He served as the TPC co-chair for a number of IEEE conferences and workshops. He has served as an Associate Editor for IEEE Transactions on Dependable and Secure Computing, IEEE Transactions on Services Computing, IEEE Internet of Things Journal, IEEE Networking Letters, and the Journal of Blockchain Research.



Xiaohua Jia (Fellow, IEEE) received the BSc and MEng degrees in 1984 and 1987, respectively, from the University of Science and Technology of China and the DSc degree in 1991 in information science from the University of Tokyo. He is currently the chair professor with the Department of Computer Science at the City University of Hong Kong. His research interests include cloud computing and distributed systems, computer networks, wireless sensor networks, and mobile wireless networks. He is an editor of the *IEEE Transactions on Parallel and Distributed Systems (2006-2009), Wireless Networks, Journal of World Wide Web, Journal of Combinatorial Optimization, etc.* He is the general chair of ACM MobiHoc 2008, TPC co-chair of IEEE MASS 2009, area-chair of IEEE INFOCOM 2010, TPC co-chair of IEEE GlobeCom 2010-Ad Hoc and Sensor Networking Symposium, and Panel co-chair of IEEE INFOCOM 2011. He is a fellow of the IEEE Computer Society.