Fountain: DAG-Based Separate BFT Consensus Made Hashgraph Practical

Hao Yin, Changling Zhou, Weiping Wen, and Yiwei Liu

***Abstract*—The limited transaction throughput performance is the primary challenge for single-chain structure blockchain in practical applications. Directed acyclic graph (DAG) technology offers a novel topological structure, significantly improving blockchain's ability to process massive transactions by containing numerous blocks in a parallel way. However, the existing parallel-chain schemes embed DAG into the Practical Byzantine Fault Tolerance (PBFT) protocol stages to expand the instance, which requires an extent of overall synchronization between multiple chains. Overall consensus synchronously among all parallel chains may delay the system's progress, particularly when the graph structure is unevenly distributed. This paper introduces a new DAG-based blockchain named Fountain, which aims to provide a loosely coupled consensus algorithm based on the classic parallel-chain scheme Hashgraph. The Fountain scheme changes consensus finality from an overall behavior to a separate behavior, thereby reducing block commit latency from the perspective of each parallel chain. Theoretical analysis reveals that our scheme maintains the same security level as Hashgraph, while experimental simulations demonstrate its effectiveness in optimizing commit latency.**

***Index Terms*—Blockchain, parallel chains, hashgraph, consensus algorithm, low latency.**

I. Introduction

BLOCKCHAIN [1] is a novel distributed ledger technology using a completely peer-to-peer approach, creating a trustless transaction processing platform. This technology enables efficient information exchange and data sharing. Consequently, blockchain has attracted significant research attention from academia and industry, finding applications in finance, logistics, healthcare, supply chains, and more. Bitcoin [2] serves as a classic example of blockchain technology implementation, utilizing a single chain structure built using hash functions and engaging in competitive mining based on proof of work. Both chain structure and competitive mining constitute the Nakamoto consensus mechanism [3], [4] to reach consistency. However, these systems often fall short in meeting practical business performance needs. For instance, Bitcoin processes only 5-7 transactions per second [5], prompting extensive research into enhancing blockchain transaction throughput.

Initial research primarily focused on innovating Nakamoto consensus protocols, while maintaining the traditional chain structure for organizing historical blocks. Some approaches, like Proof of Stake [6], [7], Permacoin [8], and Bitcoin-NG [9], focus on maximizing computing power utilization to enhance performance. These excellent schemes significantly expand the research breadth of the blockchain consensus mechanisms. Other studies integrate Byzantine Fault Tolerant (BFT) algorithms, like Sharding [10], Algorand [11], and Hotstuff [12], to accelerate block confirmation, thereby replacing traditional mining protocols. These ingenious schemes illustrate more possibilities from another track in designing blockchain consensus mechanisms. Although they have made effective improvements in consensus, the chain structure cannot simultaneously process enormous blocks and thus still faces performance bottlenecks.

Directed acyclic graphs (DAG) enable blockchains [13], [14] to process blocks in parallel by restructuring their organization. This approach is a highly efficient scaling method that significantly enhances system performance. Constructing a blockchain system with a graph structure typically involves two steps. Firstly, except for the genesis blocks, the system allows blocks to reference multiple preceding blocks, thereby forming a DAG based on predetermined block reference strategies. Secondly, the system implements the designed consensus protocol to decide finalized blocks, thus forming a subgraph. Within it, the system identifies a pivot chain and calculates the total ordering based on round partition or time.

According to the graph structure, DAG-based blockchain (also named blockDAG) can be classified into three types [15], [16]: convergence, divergence, and parallel. The convergence-type blockDAG specifies a pivot chain for global ordering, and the generated extra blocks are contained in the pivot chain as much as possible. Although these additional blocks are the increased throughput of the system, the performance improvement is limited. The divergence-type blockDAG allows any number of references, resulting in an irregular graph structure. Consequently, this leads to a high complexity in the consensus algorithm and weak reliability

Received 5 February 2025; revised 30 May 2025; accepted 30 May 2025. Date of publication 3 June 2025; date of current version 7 October 2025. The associate editor coordinating the review of this article and approving it for publication was A. Veneris. *(Corresponding author: Hao Yin.)*

Hao Yin is with the Research Center of Cyberspace Security, Peking University Changsha Institute for Computing and Digital Economy, Changsha 410205, China (e-mail: yinhao@icode.pku.edu.cn).

Changling Zhou is with the Research Center of Cyberspace Security, Peking University Changsha Institute for Computing and Digital Economy, Changsha 410205, China, and also with the Computer Center, Peking University, Beijing 100871, China (e-mail: zclfly@pku.edu.cn).

Weiping Wen is with the Research Center of Cyberspace Security, Peking University Changsha Institute for Computing and Digital Economy, Changsha 410205, China, and also with the School of Software and Microelectronics, Peking University, Beijing 100871, China (e-mail: weipingwen@pku.edu.cn).

Yiwei Liu is with the Cybersecurity Department, Defence Industry Secrecy Examination and Certification Center, Beijing 100089, China (e-mail: yiweiliu_disecc@163.com).

Digital Object Identifier 10.1109/TNSM.2025.3576128

in the ordering results. The parallel-type blockDAG organizes blocks using multiple chains and inter-chain references, achieving balanced performance in practical applications. Compared to the convergence-type blockDAG, it processes more blocks in parallel. While compared to the divergence-type blockDAG, it enables simpler ordering rules. Besides, the architecture of aggregating several parallel chains makes it convenient to supervise.

A. Challenges and Motivations

Currently, some DAG-based blockchains adopt PBFT protocols to accelerate consensus, but existing schemes use message communication to design a consensus algorithm, while the DAG topology is introduced to construct multiple instances for parallel execution. On the one hand, it makes the system more complicated to implement. On the other hand, it requires synchronization of multiple consensus instances, which causes tightly coupled to present weaker robustness.

Hashgraph [17] is a classic parallel-type blockDAG consensus algorithm used to build a public stablecoin platform called Hedera [18]. Such a platform provides public blockchain service, where the core consensus algorithm Hashgraph runs on fixed numbers of parallel chains maintained by different nodes. Nodes broadcast blocks and build DAG to make consensus, instead of relying on message transmission. Compared to state-of-the-art schemes, Hashgraph is adapted to the blockchain network environment with dynamics and scalability properties. However, the consensus algorithm designed in Hashgraph still faces the practical technical challenge of binding all parallel chains together, even if one is already confirmed in certain rounds. The confirmed chain has to wait for other chains to reach consistent in aligned rounds, where the extra waiting time is considered redundant consensus latency. As a result, the motivation of the paper is: ***decoupling the parallel chains to reduce consensus latency for better responsiveness.***

B. Contributions

The parallel chains are decoupled to reach finality separately, where each chain is a stream of water spreading outward like a fountain. Thus, we introduce the proposed Fountain scheme to address the previously mentioned challenges and optimize the Hashgraph scheme. The concrete contributions of this paper are summarized as follows:

- We propose a distinct consensus mechanism for parallel-type blockDAG, where a block can only be committed by subsequent rounds of blocks in the same parallel chain. This approach effectively reduces consensus latency by decoupling parallel chains.
- We design several block reference rules to optimize graph connectivity and provide a chain fork solution to ensure system liveness. The rules can guide fast convergence and stabilize consensus.
- We theoretically analyze the Fountain scheme's safety and liveness, which results in the same security level as the hashgraph scheme. The experimental simulation results demonstrate that the proposed separate consensus indeed enhances responsiveness.

C. Organization

We organize the rest of the paper as follows. The system model and graph topological structure are presented in Section II. Section III shows the strategies to generate a graph, illustrates the separate consensus scheme, and provides a fork solution. We make the theoretical analysis in Section IV and take the experimental simulations in Section V. Section VI introduces the related work and compares the differences. Finally, we conclude this paper in Section VII.

II. BlockDAG System

This section introduces the system model, also including assumptions and required properties. Subsequently, we present the basic graph topological structure of a blockDAG to help intuitive understanding. Additionally, it is necessary to explain the main concepts in Hashgraph on which our optimization work is based.

A. System Model

We build a permissioned blockDAG system in which participants are required to be authorized as consensus committee members. The members batch transactions into blocks based on specific time intervals or a maximum block capacity. New blocks are transmitted to other members using a gossip protocol [19], [20] to minimize communication costs. The system uses DAG with multiple parallel chains to organize blocks, which can execute multiple BFT instances simultaneously, thus maximizing the system's capacity.

Assumptions. Let the system contain n participants, we assume that it has up to f malicious members, each of whom can make arbitrary faults (i.e., Byzantine faults), such as losing connection, not responding, publishing ambiguous messages, etc. Additionally, these f malicious members may collude or be controlled by a single adversary. We require that the number of corrupted members in the system is less than $n/3$, i.e., $f < n/3$. Besides, we assume a partially synchronous network in the system like Hashgraph, with a maximum latency of block transmission, denoted as Δ. This ensures that blocks published by honest members are always receivable by other honest members. The system possesses a Global Stabilization Time (GST), after which the network achieves synchronization.

A blockDAG system reaches a consensus that is similar to achieving consistency in a distributed system, and it ought to satisfy the following properties.

- *Safety*. In any parallel chain, no two participants commit different blocks at the same index location.
- *Liveness*. Any participants always commit new blocks on the parallel chain, enlarging the set of decided blocks.

B. Graph Topological Structure

Our system employs a graph structure akin to Hashgraph, where each generated block (except the genesis block) must contain two hash references. One reference is to a "father" block, created by the same participant. The other is to an "uncle" block, created by a different participant. Starting from scratch, we assume a consensus committee consisting of four

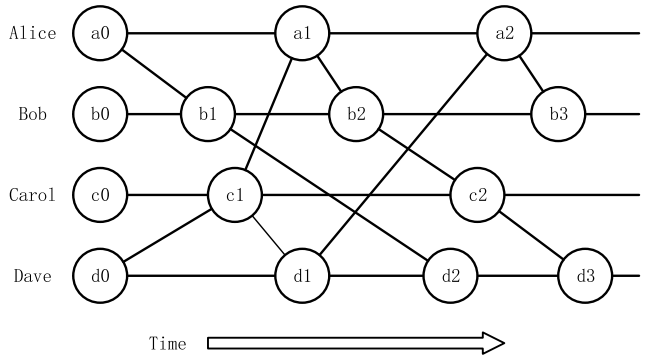

Fig. 1. An example of directed acyclic graph in the system.

TABLE I
SYMBOL DENOTATION

Symbol	Description
i	The index of participants, also the parallel chains
$chain_i$	A list of blocks created by participant i
DAG	A set of blocks forming the graph
b, u, v	The blocks in the graph
r	The round label assigning to blocks
w	A witness block, the first block in each round

Algorithm 1 Basic Utilities

procedure *index*(i, b) ▷ Get the index of block b
  if $\exists x$ s.t. $chain_i[x] = b$ **then**
    return x
  return $\perp$
procedure *dag*(b) ▷ Get the subgraph from block b
  if $b \notin DAG$ **then return** $\perp$
  $s \leftarrow set()$
  $q \leftarrow [b]$ ▷ q is a queue with *pop* and *push* methods
  while q is not empty **do**
    $e \leftarrow q.pop()$
    $s \leftarrow s \cup \{e\}$
    if $e.father \neq \perp$ **then** $q.push(e.father)$
    if $e.uncle \neq \perp$ **then** $q.push(e.uncle)$
  return s
procedure *path*(u, v) ▷ Check if u see v in the graph
  if $u \notin DAG$ or $v \notin DAG$ **then return** $\perp$
  if $\exists\{b_1, b_2, ..., b_k\}$ s.t. $b_1 = v, b_k = u$, and $\forall i \in [2..k]: b_{i-1} \in (b_i.father \cup b_i.uncle)$ **then**
    return 1
  return 0

participants, each of them possessing a unique genesis block. The committee members run two loop programs: the *sync* program receives new blocks and adds them to the graph, and the *pack* program creates new blocks and gossips them to other members. Each newly created block always references the latest block in the current graph view. We demonstrate such a DAG in Fig. 1.

The "father" references form a unidirectional hash chain, and we denote $chain_i$ as the sequence of blocks created by participant p_i. Meanwhile, the "uncle" references are used to link multiple chains, and we denote *DAG* as the set of blocks belonging to the graph. Note that each participant can only create blocks located on their chain. They cannot forge blocks on other chains due to the non-repudiation of cryptographic signatures. Besides, the inevitable network delays result in participants having different local views of the blockDAG.

The commonly used symbols in this paper are shown in Table I. We designed a block data structure as follows, which is simplified and contains the necessary fields.

```
struct block {
    time,      // the timestamp of creating block
    nonce,     // the random number padding
    father,    // the referred block from itself chain
    uncle,     // the referred block from other chains
    creator,   // the public identity of block creator
    signature} // the signature of block content
```

The basic utilities are specified in Algorithm 1. *index*(*i*, *b*) procedure gets the unique index number of block *b* in chain *i*. For example in Fig. 1, block the index number of block $c1$ in Carol's chain is 1. *dag*(*b*) procedure returns a subgraph seen by block *b*. For block $a1$, its subgraph from block $a1$ is *dag*($a1$) = {$a0$, $c0$, $d0$, $c1$, $a1$}. *path*(*u*, *v*) procedure check if exist a reference path from block *u* to block *v* in the DAG. We can easily get the fact that *path*($d1$, $b1$) = 0 and *path*($d2$, $b1$) = 1.

C. Concepts in Hashgraph

Here we need to explain some key concepts in Hashgraph because our scheme also uses them to design the consensus algorithm. These concepts are defined progressively and set similar requirements to PBFT.

- *See:* Given any two blocks *x* and *y*, *x* can *see* *y* means a path from *x* to *y* in the DAG.
- *Strongly see:* Given any two blocks *x* and *y*, if *x* can see a set of blocks *Z*, each of which $z \in Z$ from distinct parallel chains, satisfying the scale of set *Z* is larger than $2n/3$ while each *z* can see *y*, thus *x* can strongly see *y*.
- *Witness: Strongly see* divides the blocks *x* and *y* into different rounds, and the first *x* that can *strongly see* a set of blocks *Y*, each of which $y \in Y$ from distinct parallel chains, satisfying the scale of set *Y* is larger than $2n/3$, thus *x* is defined as a witness at that round.

Besides, the Hashgraph scheme defines more complex concepts based on those, such as famous witnesses, unique witnesses, etc., which are used to align different parallel chains to design consensus algorithms. The proposed Fountain scheme purges these additional definitions and achieves separate consensus on the parallel chains.

III. FOUNTAIN FOR SEPARATE CONSENSUS

In this section, we demonstrate the block reference strategies adopted by the participants to generate a DAG. We then present the core method enabling participants to reach local consensus using these reference relationships. Lastly, we

provide a chain solution to address the liveness problem in the event of chain forks.

A. Graph Generation

The Hashgraph [17] scheme presents an idea of "gossip about gossip", which lets an uncle reference reflect the message transmission path. On receiving a block from others, participants correspondingly create a new block to refer to and transmit the new block to others. This passive block generation can prevent repeat reference, but it is vulnerable to DDoS attacks in a real network since the participant may receive lots of blocks at the same time.

In this paper, we attempt two active block reference strategies by decoupling block generation from block reception. Intuitively, the uncle reference should guide the participants to learn more about the graph. Tighter relations between multiple chains result in a stronger consensus.

The first intuitive strategy is that: *the newly created block takes priority for referring to the block with higher heights in other chains*. We call it as height strategy and define the *height* of a block as the maximum number of blocks traversing from the designated block to the genesis blocks. Let the *height* of the genesis block be 0, and the *height* of block b can be formalized as follows.

$$height(b) = max(height(b.father), height(b.uncle)) + 1$$

The height strategy can rapidly synchronize participants' knowledge about the graph. Considering a general case that the speeds of block generation are different across multiple chains. For the parallel chains with slow block generation speed, their latest blocks may be frequently referenced.

Therefore, the second strategy is that: *the newly created block takes priority for referring to the block with greater view differences in other chains*. We call it as differ strategy and define the *differ* of the referred uncle block as the number of blocks that can be accessed by the uncle block but cannot be accessed by the father block. Let a new block with father block u and uncle block v, the *differ* of the new block can be formalized as follows.

$$differ_u(v) = |dag(v) - dag(u)|$$

The differ strategy compares the difference of information among other views, which guides the participants in filling up the missing blocks in their views as soon as possible. The block known to a chain has little possibility of repeated reference by the same chain, which reduces redundant communication costs.

B. Fountain Consensus

The consensus of the blockDAG system uses the block reference information to locally finalize consensus without extra message communication. In the DAG with multiple parallel chains, the consensus algorithm has roughly three steps: 1) **dividing rounds**. The first step is to assign a self-increasing round label for each block, which aligns the progress of parallel chains. 2) **committing blocks**. The second step is to find the rounds that complete consistent confirmation, based on the reference relations in the DAG. 3) **crossing sort**. The third step is to decide the ordering of blocks from different parallel chains.

Algorithm 2 Assigning Round Label to Blocks

```
1:  procedure round(b)
2:      if b is a genesis block then return 0
3:      r ← max(round(b.father), round(b.uncle))
4:      see ← {argmax_{index(i,u)}{u ∈ chain_i s.t. path(b, u) = 1}}_{i∈{1,2,...,n}}
5:      strongly ← {argmax_{index(j,v)}{v ∈ chain_j s.t. ∃s ⊂ see, |s| > 2n/3, ∀g ∈ s, path(g, v) = 1}}_{j∈{1,2,...,n}}
6:      if |{d ∈ strongly s.t. round(d) = r}| > 2n/3 then
7:          return r + 1
8:      return r
9:  procedure witness(b)
10:     if round(b) > round(b.father) then
11:         return 1
12:     return 0
```

1) Dividing Rounds: Similarly, we use the two notions of *see* and *strongly see* to determine rounds. Block u can *see* block v means that there is a path from u to v in the DAG, i.e., *path*(u, v) = 1. There is a set of blocks from different parallel chains seen by block u, in which each block can see block v. If the size of the set is more than 2n/3 then we say u can *strongly see* v. Note that the action of *see* is transitive, if block a can see block b and block b can see block c, then block a can see block c. This property is also adapted to *strongly see* because it is defined based on *see*.

We use *round* procedure to recursively compute the round number of block b as shown in Algorithm 2. In line 2, we set the round of genesis blocks to 0. The set *see* contains the blocks seen by block b from different parallel chains, in which selecting the block with the maximum index, as shown in line 4. Next in line 5, the set *strongly* contains the blocks strongly seen by block b from different parallel chains, also choosing the block with the maximum index. In lines 6-8, if block b can strongly see more than 2n/3 blocks in round r, then block b is into round $r + 1$, where r is the maximum round number that block b can see, as shown in line 2.

We demonstrate the *round* procedure in Fig. 2. The purple blocks from different parallel chains can be seen by the green block, which consists of a set of blocks (corresponding to the set *see* in Algorithm 2). The green block strongly sees the red block through the set, where the size of the set is more than 2n/3. There are more than 2n/3 red blocks in round r strongly seen by the green block, and thus it enters round $r + 1$. Additionally, the first block in each round is marked as a witness block, which means if *round*(b) > *round*(b.*father*) then block b is the witness block in the new round. We use *witness* procedure to check the property in lines 9-12. As shown in Fig. 2, the green block is the witness block in round $r + 1$ for Alice. Without loss of generality, all genesis blocks are the witness blocks in round 0 for participants.

2) Committing Blocks: Witness blocks represent the start of rounds and can be used to align the progress of different

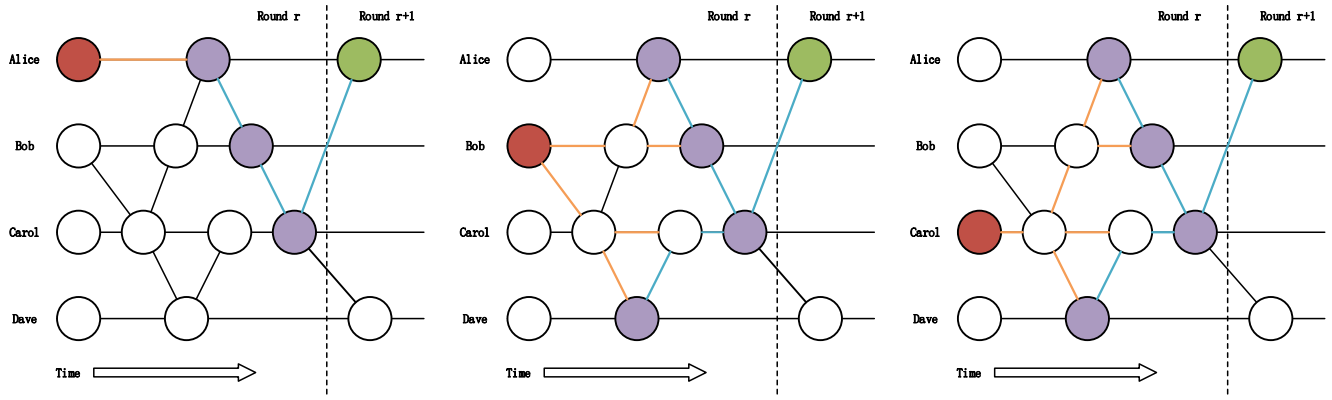

Fig. 2. The process of dividing rounds in the DAG via the notion of *strongly see*.

Algorithm 3 Finding a Decided Block For Consensus

```
1: procedure decideby(w)
2:     if witness(w) = 0 then return ⊥
3:     r ← round(w)
4:     c ← w.creator
5:     max_r ← the max round in chain_c
6:     for i in [r + 2..max_r] do
7:         v ∈ chain_c s.t. round(v) = i, witness(v) = 1
8:         ss ← {u ∈ chain_j s.t. round(u) = i − 1, witness(u) = 1, path(v, u) = 1}_{j∈{1,2,...,n}}
9:         if |{u ∈ ss s.t. path(u, w) = 1}| > 2n/3 then
10:            return v
11:    return ⊥
```

parallel chains. The main difference between the Hashgraph scheme and our Fountain scheme is the approach of committing blocks. We ignore the trivial details and finalize consensus just using the witness blocks. In the simplified DAG, each round has at most one block for every participant. According to the definition of round, the block in the current round has more than $2n/3$ references to the blocks in the last round. We can reduce the reference relations of witness blocks among rounds into the message broadcast of PBFT paradigm.

Given an instance to interpret the reduction process. For witness block w in round r, the participant publishes w is equivalent to the pre-prepare phase in the PBFT protocol. As DAG grows, new witness blocks take participants to enter new rounds. If a witness block u in rounds $r' > r$ can see w, then it means that the reference from u to w accomplishes the prepare phase in the PBFT protocol because of the *strongly see* definition. That is to say, once having more than $2n/3$ witness blocks in some round r' can see w, it finishes the commit phase in the PBFT protocol, and thus w can be safely committed. Assuming there is a witness block v in the next round can see such a set of u in round r', we say w is decided by v. In our scheme, we require that the witness blocks can be decided by the witness blocks from the same parallel chain, i.e., v and w have the same creator.

Algorithm 3 outlines the *decideby* procedure, which identifies the block that decides a specified witness block w. In line 5, We first get the maximum round number of the parallel chain where w is located. Lines 6-8 then traverse the witness block v with higher rounds in the same parallel chain to find the set of witness blocks that can be seen in the last round. In lines 9-10, if such a set includes more than $2n/3$ witness blocks that can see w, then we get the decide block v. Due to the transitive feature of 'see' and 'strongly see', the fact that v decides w can deduce two natural results: In the same parallel chain, i) the predecessor blocks of w can be still decided by v, and ii) the successor blocks of v can always decide w. As shown in Algorithm 3, the *decideby* procedure outputs a witness block with the smallest index on the parallel chain that decides the given witness block.

The principle of committing blocks in the simplified DAG is illustrated in Fig. 3. Each block is the first block in rounds of a parallel chain, i.e., witness blocks. The simplified DAG with only witness blocks has a similar topological structure to the DAG-Rider [21] and Bullshark [22] schemes. However, we do not adopt the approach of those schemes where all parallel chains reach consensus as a whole in a specific round. Our scheme commits a block on the current parallel chain with assistance from other parallel chains. For instance, the blue block v decides the green block w, as it sees the latter through more than $2n/3$ purple blocks in the last round. Similarly, the blue block e decides the green block o, but this decision is delayed by one round.

3) Crossing Sort: When a witness block is decided, it means that all blocks before the round of the witness block have completed the PBFT protocol and can be safely submitted. In a single parallel chain, the block reference and path procedure naturally create a total ordering for all blocks belonging to that chain. To sort any two blocks located at different parallel chains, we execute a predefined deterministic algorithm on the subgraph of DAG to get the ordering results across chains. Based on the alignment property of rounds, we identify the rounds with the ability of crossing sort, in which the corresponding witness block on all parallel chains has been decided. The consensus round can be formalized as follows.

$$consensus = \{r \mid round(w) = r, decideby(w) \neq \perp, \\ \forall w \in chain_i, i \in \{1, 2, \ldots, n\}\}$$

Only when some round is included in the *consensus* set can the blocks of that round be ordered across different

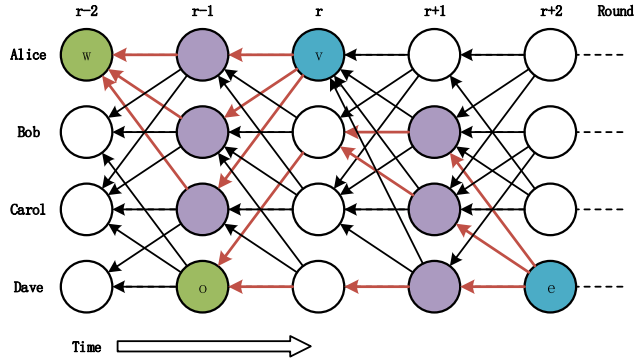

Fig. 3. The simplified DAG using witness blocks and the core principle of committing blocks.

parallel chains. Note that blocks from lower rounds are sorted before blocks from higher rounds. If two blocks are in the same round, our system uses the *findOrder* procedure in the Hashgraph scheme to output the ordering results. The blocks are sorted by timestamps received from others, and then ties are sorted by whitened signatures. We focus on the consensus scheme for a block set, where the ordering results can be obtained by specifying any deterministic algorithm for the block set. For simplicity, we refrain from providing further elaboration in this paper.

C. Fork Solution

The consensus outlined in the previous subsection relies on the premise that the parallel chain has no forks. However, when forks occur within some parallel chain, a round in the simplified DAG may have multiple witness blocks. This situation introduces ambiguous decisions in committing blocks and can even impede parallel chains from entering new rounds. To resolve forks, we provide block selection rules to select valid blocks, and honest participants should follow the rules to create new blocks to converge branches. The basic design principle of the fork selection rule is to prefer forks that have more workload. The workload here includes twofold: 1) the longer the fork lasts, the greater the workload, and 2) the greater the block difficulty at the fork point, the greater the workload. (refer to Bitcoin's proof-of-work)

In the system, we define a *fork* of the DAG as a situation where two blocks $u, v \in chain_i$, created by participant p_i, have the same father reference, i.e., $u.father = v.father$. For a parallel chain, those two blocks occupy the same position and we have $index(i, u) = index(i, v)$. Assuming a participant p_j discovers two of the latest blocks $b, d \in dag - chain_j$ originated from different parallel chains, where these blocks satisfy $path(b, u) = path(d, v) = 1$. We can state that the participant p_j encounters a *conflict* at blocks b and d. The conflict indicates the latest blocks of different branches, caused by a fork within parallel chains.

When choosing an uncle reference on different branches, the rule for participant p_i is as follows.

1) Priorly choosing the branch where the conflict point experiences more rounds from the fork point. That is to say, the participant p_i refers to block b as uncle block if the situation satisfies $round(b) - round(u) > round(d) - round(v)$.
2) Priorly choosing the branch where the block in the fork point has greater block difficulty when rule i) is not satisfied. That is to say, the participant p_i refers to block b as uncle block if the situation satisfies $hash(u) < hash(v)$.

When encountering a fork, the participants only accept blocks with higher difficulty. In another case, if a branch extended from the fork enters a new round, the participants will directly ignore or drop the forks with lower rounds.

We implement block selection rules to guide the DAG growth towards higher block difficulty. To resolve conflicts caused by forks, participants revert to the fork point in their parallel chain and replace stale blocks with new ones. In the current round, participants may identify conflicts originating from the previous round's forks and address them by altering their blocks. When entering the next round, the winning fork will gain majority support from honest participants. The rule i) tells that any other forks that occurred in the last round cannot be accepted in the next round. As a result, the remaining branches will be discarded.

IV. Theoretical Analysis

In this section, we give the theoretical analysis to prove that the proposed Fountain scheme satisfies the properties of safety and liveness for making consensus on the blockDAG system.

A. Safety Proof

In the progress of graph generation, the chain fork damages the consistency of the DAG because it may result two participants in seeing different views on one parallel chain. The definition of round and dividing blocks into self-increasing rounds has the same effect as PBFT protocols for ensuring the system's safety.

Lemma 1: If a chain fork is seen in round r, then after entering round $r + 2$ the participants cannot strongly see the chain fork.

Proof: For simplicity, we assume two witness blocks b and d in round r form a fork point on some parallel chain, and participant p_i enters round $r + 1$ by strongly seeing $2n/3$ witness blocks in round r while its view contains the fork. Due to network delays and malicious behavior, the participants may see distinct blocks on the parallel chain. Let U denote the set of participants who support fork b while V denote the set of participants who support fork d. When entering round $r + 2$ via fork b, it means that more than $2n/3$ participants support fork b, where those participants belong to set U and we have $|U| \geq 2n/3$. If now a witness block in round $r + 2$ can see fork d, then the size of set V must also be greater than $2n/3$. Remove at most $n/3$ malicious participants who make byzantine faults, there is at least one honest participant who is located in set U and move to set V. That honest participant changes its vote from fork b to fork d, where the round difference of the former branch is apparently greater than that of the latter branch. it is a contradiction according to the block reference strategy i), and thus we prove the lemma. ■

Lemma 2: Before the last two rounds of the latest round, any chain fork cannot cause participants to change their votes and the subgraph composed of all historical blocks is unique without chain forks.

Proof: By Lemma 1, after entering round $r + 2$, the chain fork before round r cannot be strongly seen. Thus, if the current latest round is r, then chain forks occurred before round $r - 2$ cannot affect the references of historical blocks. The subgraph composed of that is unique and without chain forks. ■

Theorem 1 (Safety): No two participants commit different blocks at the same index of some parallel chain.

Proof: According to Algorithm 3, a witness block is committed when there is a round with a majority of honest participants publishing blocks to strongly see it and the set consisting of those blocks is strongly seen by the same participant in the next round. Assuming the current latest round is r, the committed witness blocks may only exist before round $r - 2$. By Lemma 2, the subgraph composed of the committed blocks is unique without chain forks. In that subgraph, there is only one block at each index of any parallel chain. Meanwhile, those committed blocks have obtained more than $2n/3$ participants' confirmations, which is supported by the majority of honest participants. In summary, we prove that no two participants commit different blocks at the same index of some parallel chain. ■

B. Liveness Proof

As the graph grows, some parallel chains continue to generate new chain forks, causing the system to be unable to enter a new round. By applying the fork solution, we can mitigate the liveness problem when the system encounters a chain fork.

Lemma 3: Honest participants can always enter a new round.

Proof: There are two situations when the system faces a chain fork. One case is that a fork point has a lower block difficulty or its branch has a smaller round difference. The participants will reject the fork because it does not satisfy the block selection rule, and thus the system can enter a new round. Another case is that branches sourced from a fork have the same round difference but the new fork point has a higher block difficulty. It persuades honest participants to change their votes by generating new blocks to replace the stale ones. Whenever the malicious participant generates a fork at the same point, the time cost of computing a valid block with higher difficulty is commonly greater than that of the last fork. That is to say, the time interval of creating a new proposal at the same point is enlarged with time. Once the time interval is larger than the maximum transmission latency Δ, a branch sourced from the last fork may enter the next round when a new fork is created at the same point. The new fork cannot delay the system from generating a new round, and thus honest participants can always enter a new round. ■

Theorem 2 (Liveness): Any participants always commit new blocks on the parallel chain.

Proof: To commit a new block, the participant broadcasts the block as a proposal so that it can be confirmed by a majority of honest participants. After the maximum transmission latency or reaching a global stabilization time, all participants are aware of the newly created block. By Lemma 3, honest participants always enter a new round. Eventually, there is a round with a majority of honest participants seeing that block, and thus the participant can always commit new blocks. ■

V. Experimental Simulation

In this section, we design simulation experiments for the proposed Fountain scheme and compare it with the hashgraph scheme. We simulate the graph generation to observe the impact of block reference strategies on a graph and measure the latency of consensus decisions to evaluate performance.

A. Instructions and Settings

Hedera [18] is a DAG-based blockchain system running on the public network. It implements the core algorithm of the hashgraph scheme and provides various SDKs to build stablecoin applications. Unfortunately, the official does not open source the code of the hashgraph algorithm. By searching related projects in GitHub, we found a project named py-swirld that implements the hashgraph algorithm in Python language. This project simulates the process of generating a graph and visualizes the progress of executing the consensus. We forked the py-swirld project and modified the program logic of the key part. The code is open source in the public repository,[1] and the branch hashgraph and fountain write the corresponding experiment scripts.

The simulation experiments are conducted on a laptop that runs Windows 11 with Intel Core i7-1260P CPU and 32GB RAM. We let each parallel chain generate roughly 1000 blocks and conduct 100 times experiments to get average results. Since the system follows a byzantine fault tolerance (BFT) configuration, we set the scale of parallel chains to $n = 3f + 1$ and let f be from 1 to 7. We also tried the parallel chains scale of more than 22, but from the simulation results, the overall latency of both schemes tends to be stable. The reason is that more parallel chains lead to better connectivity. Therefore, we refer to the EOS Network,[2] which only sets 21 super nodes as the core to generate blocks. The simulation experiment uses this as the upper limit for comparison, illustrating that our proposed solution achieves better results. In the experiments, we use rounds as the basic unit to analyze results instead of concrete running time. The whole simulated network is fully connected. When a node acts, it selects another node according to the scheme strategy, thus linking two parallel chains through block references.

B. Graph Generation Simulation

The logic for creating new blocks in the py-swirld project follows the hashgraph scheme. First, the main program randomly specifies a chain for building a new block. Then, the

[1] https://github.com/ug1y/py-swirld
[2] https://eosnetwork.com

TABLE II
THE IMPACT OF DIFFERENT GENERATION MODES ON GRAPH STRUCTURE

Generation Modes		Scale of Parallel Chains						
		4	7	10	13	16	19	22
Max round	gossip mode	**279.8**	187.9	**159.0**	**141.5**	131.4	**124.0**	**119.1**
	height mode	265.8	167.1	136.3	118.5	108.2	102.3	96.5
	differ mode	279.1	**189.3**	158.3	141.4	**131.5**	**124.0**	119.0
	random mode	145.1	68.9	48.7	38.7	32.2	28.1	25.0
In-degree std	gossip mode	1.411	1.421	1.410	1.409	1.414	1.415	1.406
	height mode	1.380	1.478	1.516	1.554	1.578	1.588	1.605
	differ mode	**1.345**	**1.347**	**1.349**	**1.354**	**1.355**	**1.361**	**1.360**
	random mode	1.738	1.847	1.898	1.919	1.941	1.954	1.969

chain randomly selects a chain in the rest to sync the graph information and refer to the latest block of it. The above process simulates the gossip communication protocol and we call it gossip mode in graph generation. To apply the active block reference strategies, we split the sync and refer stages to two functions. The specified chain randomly selects a chain to sync but refers to a block according to the given block reference strategies, and thus we have height mode and differ mode in graph generation. Besides, we also design random mode, which allows the specified chain to randomly choose a chain to refer to. This mode does not involve any block reference strategies and thus it can be used for benchmark comparison against other modes.

As shown in Table II, we adopt the mentioned 4 generation modes to get random graphs and calculate two indicators of max round number and in-degree standard deviation. The max round is the max assigned round number of the latest block in all parallel chains. Since we give the relatively same amount of blocks for each parallel chain, the bigger the max round means that each round contains fewer blocks and better connectivity between parallel chains. The in-degree distribution can measure whether new blocks quickly spread, which is calculated by standard deviation. The smaller the value, the more efficient the propagation. And it also benefits reaching great connectivity.

As the scale of parallel chains grows, all generation modes cause fewer rounds in the graph but differ mode still maintains a lower in-degree std. It shows that differ mode can achieve great connectivity for parallel chains in the graph. From the vertical comparison, differ mode we proposed reaches optimal in graph generation, which is better than gossip mode in the hashgraph scheme, having a more uniform in-degree distribution between parallel chains.

C. Consensus Decision Latency

Compared to the hashgraph scheme, the main difference of the proposed Fountain scheme is the way of consensus decision. We decouple the parallel chains to decide consensus by themselves but reach the same ability of byzantine fault tolerance as that of the hashgraph scheme. The participant can safely commit blocks at the local parallel chain, and it crossly orders blocks with other parallel chains after the aligned rounds all reach a consensus. For the latter, both schemes adopt the same execution logic, while for the former, the proposed Fountain scheme gives an extra feature to reduce the waiting time for consensus.

In the experiments, we use the number of rounds to measure the waiting time for reaching a consensus. If block v in round i decides block w in round j, then the consensus latency for block w is calculated by $i-j$. For one simulation, we focus on the maximum consensus latency in all blocks and record that of locally committing blocks and globally crossing sort separately. For the hashgraph scheme, locally commit a block if and only if the round of that block gets a consensus decision in the global scope. Therefore, we specify one parallel chain to get the number of waiting rounds as the local consensus latency, while outputing the maximum waiting rounds among all parallel chains as the global consensus latency.

We illustrate the results in Fig. 4, which shows that the proposed Fountain scheme performs less consensus latency in both two cases. As the scale of parallel chains increases, the consensus latency correspondingly decreases. This situation is because the bigger scale of parallel chains causes more blocks in one round and results in better connectivity within that round. Once the connectivity is nearly optimal, executing consensus is equivalent to reaching consistency under the synchronous network model. Since the proposed Fountain scheme adopts the same consensus principle as the hashgraph scheme, the consensus latency of both schemes eventually tends to be significantly reduced.

VI. RELATED WORK

BlockDAG is currently a pretty frontier research direction of blockchain technology. There are a lot of works that have proposed very brilliant ideas. In this paper, we focus on the DAG paradigms that use multiple parallel chains to organize blocks and present the mainstream schemes.

Hashgraph [17] creates several parallel chains maintained by miners to record historical events. The references between events reflect the propagation path of gossip communication. The consensus is accomplished from the graph through a three-stage procedure, achieving byzantine fault tolerance guarantees. OHIE [23] allocates a block into an individual chain based on the identity computed by its block hash. Additionally, the block also refers to another chain used to specify the rank, where blocks are ordered across chains

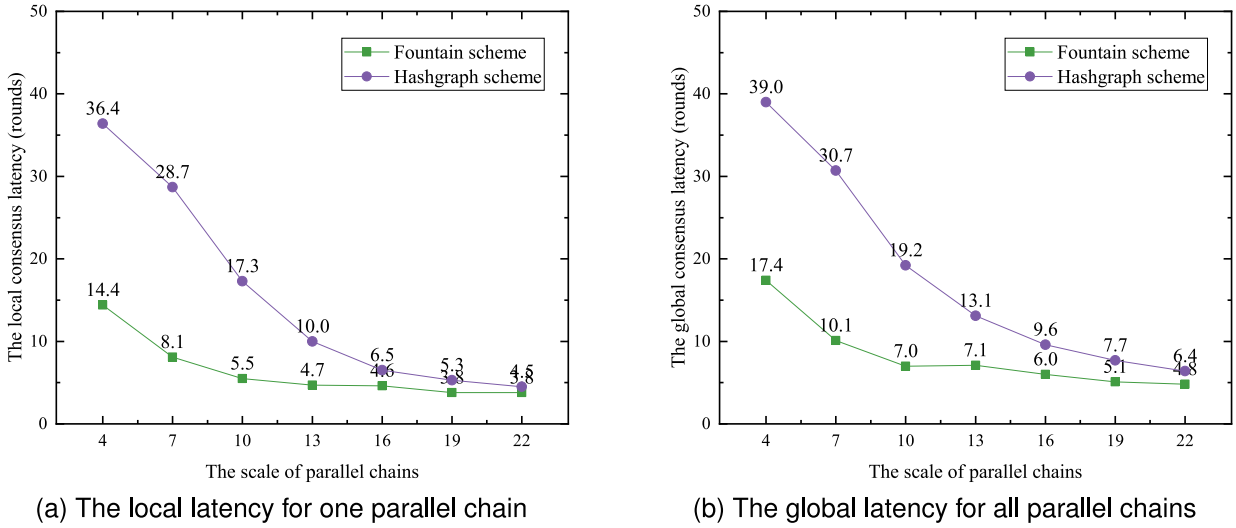

Fig. 4. The experimental results for locally committing blocks (a) and globally crossing sort (b) to compare the consensus latency between the Fountain scheme and the Hashgraph scheme.

TABLE III
THE ROUGH COMPARISON OF RELATED WORKS

	Core consensus mechanism	Parallel chains organization	Block ordering principle	Do parallel chains wait synchronously?	Are blocks divided into rounds?
Hashgraph [17]	PBFT	Members hold	Received time	Need	Almost yes
OHIE [23],Ladder [24]	Nakamoto	Random allocated	Specified rank	Not need	Almost yes
Blockmania [25]	PBFT	Members hold	Processed order	Not need	No
Prism [26]	Nakamoto	Function allocated	Blocks vote	Not need	No
Tusk [27], DAG-Rider [21], Bullshark [22], GradedDAG [28], Wahoo [29]	PBFT	Members hold	Path in graph	Need	Strict yes

via the assigned rank. Ladder [24] adopts a structured twin-chain DAG architecture with a convergence mechanism. This approach optimizes parallel block processing by designating specific convergence nodes (handling lower-chain) to handle block sorting and transaction confirmation (processing upper-chain), thereby reducing computational overhead and enhancing security. Blockmania [25] executes a practical byzantine fault tolerance protocol in parallel chains. Each node can only propose one block for the given position without any equivocation, and the created blocks jointly form a graph. Prism [26] decouples the functionalities of Nakamoto consensus into multiple chains. Blocks are divided into a transaction block, voter block, and proposer block, where the three types are relatively independent but also interactive. Narwhal [27] designed a DAG-based mempool protocol and worked with Tusk [27], an efficient BFT consensus, to optimize the state-of-the-art Hotstuff [12] scheme. It seems like extending the chain-based BFT protocols into parallels, and finding the leader blocks in DAG to reach finality. DAG-Rider [21] requires that each created block contains more than $2n/3$ blocks in the last round, thus forming a structured DAG by rounds. The first stage generates the DAG via reliable broadcast and the second stage observes the DAG and totally order with no extra communication. Bullshark [22] exploits synchronous periods to provide a practical low latency fast-path method and deprecates the need for a complex view-change mechanism. This protocol achieves all the desired properties of DAG-Rider and also introduces a partially synchronous version. GradedDAG [28] proposed a graded reliable broadcast, which together with the consistent broadcast constitutes a wave from rounds in DAG. It simplifies the rounds to broadcast data like DAG-Rider and Bullshark, etc., thus reducing latency. Wahoo [29] further optimized the reliable broadcast based on Graded DOG, presenting the provable broadcast and the enhanced provable broadcast protocols. This scheme lets one wave in DAG only contain two rounds, alternately executing those broadcast protocols.

To clearly understand the similarities and differences between related works, we make a rough comparison in Table III. The core consensus mechanism implies that the scheme relies on which consensus algorithm to process blocks. Meanwhile, the parallel chains might be held by fixed members or float allocated. The block ordering principle presents the different ways of computing block total ordering. Some schemes still need to synchronize multiple parallel chains in extending new blocks so that dividing blocks into self-increasing rounds. Our work optimizes the synchronization requirements between parallel chains in Hashgraph to make it practical. The proposed scheme decouples parallel chains to

achieve separate consensus and reduce consensus latency. On the one hand, such a scheme can support the dynamic joining and leaving of members in consortium blockchain scenarios for robust scalability. On the other hand, this scheme can support transaction isolation by business and realize cross-chain mutual authentication for strong security.

VII. Conclusion

In this paper, we proposed a separate consensus paradigm in the parallel-type blockDAG, where the parallel chains are decoupled in deciding consensus but follow the same BFT configuration. This design significantly reduces waiting latency for consensus decisions, thereby enhancing system responsiveness. Additionally, we presented block reference strategies to optimize graph generation, prioritizing the latest blocks in parallel chains using height and differ strategies. These strategies aid in adjusting the graph structure to enhance connectivity. Building on this, we provided a fork solution to address the liveness issues arising from chain forks. Participants retrace and prune the graph following the predefined block selection rules. Theoretical analysis indicates that the proposed scheme maintains safety and achieves liveness, even with chain forks. Experimental simulations reveal that the block reference strategies optimize graph connectivity and the separate consensus effectively enhances responsiveness.

References

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

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," in *Proc. Decent. Bus. Rev.*, 2008, Art. no. 21260.

[3] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou, "A survey of distributed consensus protocols for blockchain networks," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 1432–1465, 2nd Quart., 2020.

[4] H. Yin, Y. Wei, Y. Li, L. Zhu, J. Shi, and K. Gai, "Consensus in lens of consortium blockchain: An empirical study," in *Proc. Int. Conf. Algorithms Archit. Parallel Process.*, 2020, pp. 282–296.

[5] J. Göbel and A. E. Krzesinski, "Increased block size and Bitcoin blockchain dynamics," in *Proc. 27th Int. Telecommun. Netw. Appl. Conf. (ITNAC)*, 2017, pp. 1–6.

[6] S. King and S. Nadal. "PPcoin: Peer-to-peer crypto-currency with proof-of-stake." 2012. [Online]. Available: https://peercoin.net/assets/paper/peercoin-paper.pdf

[7] F. Saleh, "Blockchain without waste: Proof-of-stake," *Rev. Financ. Stud.*, vol. 34, no. 3, pp. 1156–1190, 2021.

[8] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz, "Permacoin: Repurposing bitcoin work for data preservation," in *Proc. IEEE Symp. Security Privacy*, 2014, pp. 475–490.

[9] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "Bitcoin-NG: A scalable blockchain protocol," in *Proc. 13th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2016, pp. 45–59.

[10] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 17–30.

[11] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling Byzantine agreements for cryptocurrencies," in *Proc. 26th Symp. Oper. Syst. Princ.*, 2017, pp. 51–68.

[12] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "HotStuff: BFT consensus with linearity and responsiveness," in *Proc. ACM Symp. Princ. Distrib. Comput.*, 2019, pp. 347–356.

[13] X. Lu, C. Jiang, and P. Wang, "A survey on consensus algorithms of blockchain based on DAG," in *Proc. 6th Blockchain Internet Things Conf.*, 2024, pp. 50–58.

[14] M. Raikwar, N. Polyanskii, and S. Müller, "SoK: DAG-based consensus protocols," in *Proc. IEEE Int. Conf. Blockchain Cryptocurr. (ICBC)*, 2024, pp. 1–18.

[15] Q. Wang, J. Yu, S. Chen, and Y. Xiang, "SoK: Diving into DAG-based blockchain systems," 2020, *arXiv:2012.06128*.

[16] Q. Wang, J. Yu, S. Chen, and Y. Xiang, "SoK: DAG-based blockchain systems," *ACM Comput. Surv.*, vol. 55, no. 12, pp. 1–38, 2023.

[17] L. Baird, "The swirlds hashgraph consensus algorithm: Fair, fast, Byzantine fault tolerance," Swirlds, Dallas, TX, USA, Rep. SWIRLDS-TR-2016-01, 2016.

[18] L. Baird, M. Harmon, and P. Madsen, "Hedera: A public hashgraph network & governing council," Hedera, Las Vegas, NV, USA, White Paper, Aug. 2020. [Online]. Available: https://files.hedera.com/hh_whitepaper_v2.2-20230918.pdf

[19] A. Demers et al., "Epidemic algorithms for replicated database maintenance," in *Proc. 6th Annu. ACM Symp. Princ. Distrib. Comput.*, 1987, pp. 1–12.

[20] G. Saldamli, C. Upadhyay, D. Jadhav, R. Shrishrimal, B. Patil, and L. Tawalbeh, "Improved gossip protocol for blockchain applications," *Clust. Comput.*, vol. 25, no. 3, pp. 1915–1926, 2022.

[21] I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman, "All you need is dag," in *Proc. ACM Symp. Princ. Distrib. Comput.*, 2021, pp. 165–175.

[22] A. Spiegelman, N. Giridharan, A. Sonnino, and L. Kokoris-Kogias, "Bullshark: DAG BFT protocols made practical," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2022, pp. 2705–2718.

[23] H. Yu, I. Nikolić, R. Hou, and P. Saxena, "OHIE: Blockchain scaling made simple," in *Proc. IEEE Symp. Security Privacy (SP)*, 2020, pp. 90–105.

[24] D. Hu et al., "Ladder: A convergence-based structured DAG blockchain for high throughput and low latency," in *Proc. 22nd USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2025, pp. 779–794.

[25] G. Danezis and D. Hrycyszyn, "Blockmania: From block DAGs to consensus," 2018, *arXiv:1809.01620*.

[26] V. Bagaria, S. Kannan, D. Tse, G. Fanti, and P. Viswanath, "Prism: Deconstructing the blockchain to approach physical limits," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 585–602.

[27] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Narwhal and tusk: A DAG-based mempool and efficient BFT consensus," in *Proc. 17th Eur. Conf. Comput. Syst.*, 2022, pp. 34–50.

[28] X. Dai, Z. Zhang, J. Xiao, J. Yue, X. Xie, and H. Jin, "GradedDAG: An asynchronous DAG-based BFT consensus with lower latency," in *Proc. 42nd Int. Symp. Rel. Distrib. Syst. (SRDS)*, 2023, pp. 107–117.

[29] X. Dai et al., "Wahoo: A DAG-based BFT consensus with low latency and low communication overhead," *IEEE Trans. Inf. Forensics Security*, vol. 19, pp. 7508–7522, 2024.

Hao Yin received the B.E. degree in information security and the M.S. degree in software engineering from the University of Science and Technology Beijing in 2015 and 2018, respectively, and the Ph.D. degree in cyberspace security from the Beijing Institute of Technology in 2022. He is currently an Assistant Researcher with the Research Center of Cyberspace Security, Peking University Changsha Institute for Computing and Digital Economy. His main research interests include applied cryptography, blockchain technology, distributed consensus, and privacy computing.

Changling Zhou received the Ph.D. degree in computer science from Peking University in 2016. He is currently a Senior Engineer with the Computer Center, Peking University, where he has been working in network management since 2003. He have conducted in-depth research on network architecture, network management, network information security, and next-generation Internet. His main research interests include computer network, machine learning, and mechanics.

Weiping Wen received the Ph.D. degree from the Institute of Software Chinese Academy of Sciences in 2004. He is currently a Professor with the School of Software and Microelectronics, Peking University. He published over 80 papers in domestic/core academic journals and international journals, more than 30 papers indexed by SCI/EI, written 2 monographs and 2 textbooks, and applied for nearly 50 patents in the field of information security, including 22 patents authorized. His main research interests include system and network security, big data and cloud security, and intelligent computing security.

Yiwei Liu received the M.S. and Ph.D. degrees from the Beijing Institute of Technology, Beijing, China, in 2017 and 2021, respectively. His main research interests include coding theory, social network, privacy protection, and radio frequency fingerprint recognition.