LightDAG: A Low-latency DAG-based BFT Consensus through Lightweight Broadcast

Xiaohai Dai*, Guanxiong Wang*, Jiang Xiao*, Zhengxuan Guo*, Rui Hao†, Xia Xie‡, and Hai Jin*
*National Engineering Research Center for Big Data Technology and System
*Services Computing Technology and System Lab, Cluster and Grid Computing Lab
*School of Computer Science and Technology, Huazhong University of Science and Technology, China
†School of Computer Science and Artificial Intelligence, Wuhan University of Technology, China
‡School of Computer Science and Technology, Hainan University, China
Email: jiangxiao@hust.edu.cn

***Abstract*—To improve the throughput of *Byzantine Fault Tolerance* (BFT) consensus protocols, the *Directed Acyclic Graph* (DAG) topology has been introduced to parallel data processing, leading to the development of DAG-based BFT consensus. However, existing DAG-based works heavily rely on *Reliable Broadcast* (RBC) protocols for block broadcasting, which introduces significant latency due to the three communication steps involved in each RBC. For instance, DAGRider, a representative DAG-based protocol, exhibits a best latency of 12 steps, considerably higher than non-DAG protocols like PBFT, which only requires 3 steps. To tackle this issue, we propose LightDAG, which replaces RBC with lightweight broadcasting protocols such as *Consistent Broadcast* (CBC) and *Plain Broadcast* (PBC). Since CBC and PBC can be implemented in two and one communication steps, respectively, LightDAG achieves low latency.**

In our proposal, we present two variants of LightDAG, namely LightDAG1 and LightDAG2, each providing a trade-off between the best latency and the expected worst latency. In LightDAG1, every block is broadcast using CBC, which exhibits a best latency of 5 steps and an expected worst latency of 14 steps. Since CBC cannot guarantee the totality property, we design a block retrieval mechanism in LightDAG1 to assist replicas in retrieving missing blocks. LightDAG2 utilizes a combination of PBC and CBC for block broadcasting, resulting in a best latency of 4 steps and an expected worst latency of $12(t+1)$ steps, where t represents the number of actual Byzantine replicas. Since a Byzantine replica may equivocate through PBC, LightDAG2 prohibits blocks from directly referencing contradictory blocks. To ensure liveness, we propose a mechanism to identify and exclude Byzantine replicas if they engage in equivocation attacks. Extensive experiments have been conducted to evaluate LightDAG, and the results demonstrate its feasibility and efficiency.

***Index Terms*—Byzantine fault tolerance, DAG, consensus, blockchain**

I. Introduction

The exponential rise of blockchain technology [1] has brought significant attention to *Byzantine Fault Tolerance* (BFT) consensus protocols [2]–[4], which facilitate data agreement among multiple replicas [5]. Traditional BFT protocols like PBFT [6] and HotStuff [7] process data sequentially, limiting protocol throughput. To address this limitation, recent advancements such as DAGRider [8], Bullshark [9], and Tusk [10] have introduced the *Directed Acyclic Graph* (DAG) topology, parallelizing data processing and improving performance [11]. These protocols are collectively referred to as DAG-based BFT protocols [12].

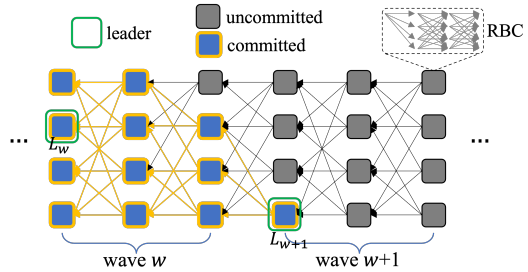


Fig. 1: Structure of existing DAG-based protocols. Blocks with orange borders are committed by the leader block L_{w+1}.

Generally, existing DAG-based (BFT) protocols progress in waves, with each wave comprising multiple rounds, as depicted in Fig. 1. In each round, every replica broadcasts a block using the *Reliable Broadcast* (RBC) protocol [13], providing the properties of consistency and totality in block delivery. In a system with $n \geq 3f+1$ replicas, a valid block must include at least $n-f$ block hashes from the previous round. A block B_1 is said to reference another block B_2 if B_1 includes B_2's hash. This reference relationship is transitive, meaning that if B_1 references B_2 and B_2 references B_3, B_1 is considered as referencing B_3 as well. Blocks referenced by a block B are referred to as B's ancestors. At the end of a wave, a block from the first round of this wave is selected as the leader block. If the leader block satisfies certain predefined conditions, such as being referenced by enough blocks, it and its ancestor blocks can be committed.

Despite the significant throughput improvements offered by DAG-based protocols, they suffer from high latency due to the RBC protocol. To be more specific, the RBC protocol requires a minimum of three communication steps (as shown in Fig. 1). Consequently, a DAG-based protocol with four RBC rounds in a wave, like DAGRider [8], necessitates at least 12 communication steps to commit a leader block, which is much larger than the latency of the non-DAG protocol (e.g., PBFT, requiring only 3 communication steps).

To address latency issues in existing DAG-based protocols, we propose LightDAG, which replaces the RBC protocol with lightweight broadcasting protocols: *Plain Broadcast* (PBC)

and *Consistent Broadcast* (CBC) [14]. PBC represents the simplest broadcast process, where the broadcaster transmits data to each replica, and each replica delivers the data once receiving it. While PBC does not guarantee consistency or totality, it achieves this with a single communication step. On the other hand, CBC is a broadcast protocol stronger than PBC but weaker than RBC. Specifically, CBC ensures consistency, even in the presence of a Byzantine broadcaster, but it cannot guarantee totality. CBC can be implemented with two communication steps.

Replacing RBC with PBC or/and CBC offers a straightforward reduction in consensus latency due to the decreased number of communication steps. However, this substitution presents challenges, as CBC lacks the totality property, and PBC lacks both consistency and totality. To address these shortcomings, we introduce a block retrieval mechanism to assist replicas in obtaining missing blocks from others, mitigating the lack of totality in CBC or PBC. Furthermore, to address the absence of consistency in PBC, we implement a consistent reference mechanism to prevent a block from directly referencing contradictory blocks.

Concretely speaking, we present two variants of LightDAG: LightDAG1 and LightDAG2, offering a trade-off between the best latency and the expected worst latency. In LightDAG1, a wave consists of three CBC rounds. Since CBC cannot ensure the totality property, we leverage the block retrieval mechanism. A replica participates in the CBC process of a block B only if it has already delivered all of B's ancestor blocks. Similar to existing DAG-based protocols, LightDAG1 selects a block from the first round as the leader block of the wave. If the leader block is referenced by at least $f+1$ blocks in the second round, it and its ancestor can be committed. Consequently, LightDAG1 achieves a best latency of 6 steps and an expected worst latency of 14 steps.

In LightDAG2, a wave also consists of three rounds: a PBC round, a CBC round, and another PBC round, in that specific order. Since a Byzantine replica can equivocate in PBC rounds, LightDAG2 leverages the consistent reference mechanism to prevent a block from referencing contradictory blocks directly. Besides, a replica (p_x) cannot vote for two blocks in the same CBC round if these two blocks directly reference contradictory blocks. Instead, p_x will send the Byzantine proof to the block proposer (p_y). p_y will then repropose a block without referencing contradictory blocks, which will be voted on by p_x. In a good situation, it only takes 4 steps to commit a block. However, if a Byzantine replica equivocates in a wave, the leader block cannot be committed during that wave, thereby compromising liveness. To tackle this challenge, LightDAG2 introduces a rule that helps each replica identify and exclude Byzantine replicas in subsequent waves. By doing so, LightDAG2 achieves an expected worst latency of $12(t+1)$ steps, where t represents the number of actual Byzantine replicas.

We implement prototype systems for both variants of LightDAG and compare them to the state-of-the-art protocols through extensive experiments. The experimental results demonstrate that both two variants effectively reduce latency and improve throughput compared to existing protocols, with LightDAG2 achieving the best overall performance. Moreover, LightDAG exhibits superior scalability, showcasing higher performance even as the system scales up. Additionally, LightDAG demonstrates a higher peak throughput when the system becomes saturated.

TABLE I: Comparison between different DAG-based protocols. Latency is measured by the communication steps, and t represents the number of actual Byzantine replicas.

	Wave length	**Broadcast**	**Best latency** †	**Worst latency**
DAGRider [8]	4	RBC	12 (10)	18
Tusk [10]	3	RBC	9 (7)	21
BullShark [9]	4	RBC	6	30
LightDAG1	3	CBC	6 (5)	14
LightDAG2	3	CBC & PBC	4	$12(t+1)$§

† If neither safety nor liveness is compromised, we only need to count the first step in RBC or CBC that reveals the leader. In this way, the latency can be reduced, whose results are shown in the brackets.

§ Although LightDAG2 shows worse performance when t is large, it can exclude Byzantine replicas and prevent them from doing evil later.

II. Background & Motivation

A. BFT consensus

Byzantine Fault Tolerance (BFT) consensus protocols are utilized to reach agreements on data among a group of mutually untrusting replicas. Some of these replicas, known as Byzantine replicas, may deviate from the protocol arbitrarily, while the remaining replicas consistently adhere to the protocol and are referred to as non-faulty replicas. In the context of blockchain systems, data to be considered are referred to as blocks, which contain multiple transactions from clients. A block is considered committed if it is agreed upon by the BFT consensus and assigned a position in the replica's ledger. A consensus protocol must satisfy the following two properties:

- **Safety:** If two non-faulty replicas commit two blocks B and B' at the same position respectively, then $B = B'$.
- **Liveness:** A transaction sent by a client will be included in a committed block eventually.

B. DAG-based consensus

Traditional BFT consensus process data in a sequential manner, agreeing upon them one by one. To improve performance, the concept of *Directed Acyclic Graph* (DAG) is introduced to the consensus design, resulting in DAG-based consensus protocols. These protocols process and agree on data in a parallel manner, whose representatives include DAGRider [8], BullShark [9], and Tusk [10].

In DAG-based protocols, the consensus progresses in successive waves, as depicted in Fig. 1. Each wave consists of multiple rounds, and in each round, each replica attempts to propose a block using the *Reliable Broadcast* (RBC) protocol. The RBC protocol can be considered as an enhanced broadcast protocol, which ensures the consistency of block deliveries by different replicas in a Byzantine environment [15], whose properties include:

- **Consistency:** If two non-faulty replicas deliver blocks B and B' respectively, then B must be equal to B'.
- **Validity:** If a non-faulty broadcaster broadcasts block B, every non-faulty replica will deliver B.
- **Integrity:** Each non-faulty replica will deliver at most one block.
- **Totality:** If a non-faulty replica delivers a block B, every non-faulty replica will eventually deliver B.

Every block must include hashes of blocks from the previous round. A block B is said to directly reference a block C if B include C's hash. Additionally, a block B is said to indirectly reference D if B references C and C references D. All blocks referenced by a block, whether directly or indirectly, are referred to as its ancestors. It is important to note that a block is considered an ancestor of itself. The blocks directly referenced by a block are also named its parents. At the end of a wave, typically through messages in the last round of that wave, a block from the first round is selected as the leader block, as indicated by blocks encircled by green boxes in Fig. 1. If the leader block satisfies specific conditions, such as being referenced by more than two-thirds of the blocks in the last round (defined in DAGRider [8]), the leader and all its ancestor blocks can be committed.

Motivation. Existing DAG-based protocols focus on improving throughput by committing multiple blocks in each wave but overlook the issue of latency. They measure latency in terms of RBC rounds and claim to have a low number of RBC rounds. However, an RBC round actually consists of at least three communication steps [16], resulting in relatively high latency, as shown in Table I. In this paper, our goal is to reduce the latency of DAG-based protocols by minimizing the number of communication steps required.

III. Overview of LightDAG

A. Model & definitions

We adopt the same model as the existing DAG-based protocols. Our system consists of $n \geq 3f+1$ replicas, where up to f replicas are Byzantine and controlled by an adversary. This adversary possesses the capability to coordinate these Byzantine replicas, enabling them to engage in malicious activities. We assume an asynchronous network where the adversary can delay messages by an arbitrary but finite period. A *Public-Key Infrastructure* (PKI) is established in the system, ensuring the integrity of messages through digital signatures. We also assume a threshold-crypto infrastructure, established by *Asynchronous Distributed Key Generation* (ADKG) schemes [17], [18]. The adversary is assumed to be computationally bounded and cannot compromise the safety of the PKI or threshold-crypto infrastructure.

Our protocol advances through successive rounds, each uniquely identified by an ever-increasing number r, starting from 1. Simultaneously, these rounds are structured into waves, with each wave consisting of three rounds. As a result, a round can also be represented by a combination of two parameters: the wave number w and a sequence number e (where $e \in \{1,2,3\}$) within that wave, taking the format $\langle w, e\rangle$. We introduce the term "slot" to refer to a position in the DAG structure, denoted as $\langle r, i\rangle$, where i represents the replica index. Ancestor blocks of a block B are represented as a unique sequence $\mathcal{A}_B$, sorted first by the round number and then by the replica index of the block's proposer. We define a *prefix* relationship between two sequences, denoted as $\mathcal{S}_1 \prec \mathcal{S}_2$, indicating that $\mathcal{S}_1$ is a prefix of $\mathcal{S}_2$. The ancestor sequence satisfies the prefix relationship, meaning if $B \in \mathcal{A}_C$, then $\mathcal{A}_B \prec \mathcal{A}_C$.

B. Building blocks

1) Consistent broadcast: Consistent Broadcast (CBC) is a commonly used utility to broadcast data in a Byzantine environment. It guarantees properties of consistency, validity, and integrity [14], but it does not ensure totality [19]. Specifically, if a non-faulty replica delivers a block B using CBC, every other non-faulty replica either delivers B or does not deliver any block at all.

Compared to RBC, CBC is a more lightweight protocol that can be implemented with fewer communication steps. While RBC requires three communication steps, CBC only needs two steps [20], [21]. In the first step, named as the `VAL` step, the broadcaster sends the block B to each replica. Upon receiving B, a replica proceeds to broadcast `ECHO` messages containing B in the second step, referred to as the `ECHO` step. At the end of the `ECHO` step, if a replica receives $n-f$ or more `ECHO` messages containing B, it delivers B. Replicas other than the broadcaster are considered participants in a CBC instance. Additionally, we say a replica participates in the CBC process of a block B if it broadcasts `ECHO` messages containing B.

2) Global perfect coin: To select a leader block/slot in each wave, DAG-based consensus protocols rely on the *Global Perfect Coin* (GPC) utility [19], [22], [23]. GPC can be considered a random function with a threshold, where each replica can call the GPC function, and the output will be revealed when the number of callers reaches a threshold. The output is a random number that is identical among all replicas.

In the context of a DAG-based protocol, the GPC function can be implemented using the threshold signature scheme. Specifically, in the last round of a wave, each replica will have a partial threshold signature on the wave number included in the block. When a replica receives t blocks, it can generate a complete threshold signature. This signature is then converted into an integer number m. By taking the modulus of m and n (the total number of replicas), the remainder i corresponds to a replica index. The slot $\langle \langle w, 1\rangle, i\rangle$ is then considered the leader slot for this wave. It is important to note that in existing DAG-based protocols [8]–[10], each slot contains at most one block due to the consistency property of RBC. Hence, each leader slot corresponds to a unique leader block, and we can perceive the leader block as being directly selected through GPC in existing DAG-based protocols. To prevent the adversary from having foreknowledge of leader slots and destroying liveness, t is typically set to a value larger than $f+1$.

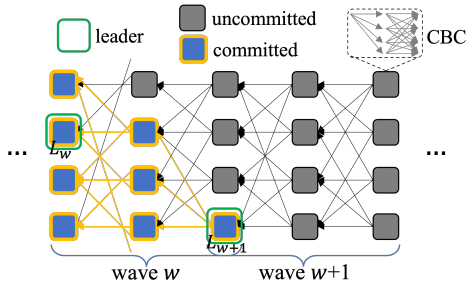


Fig. 2: Structure of LightDAG1

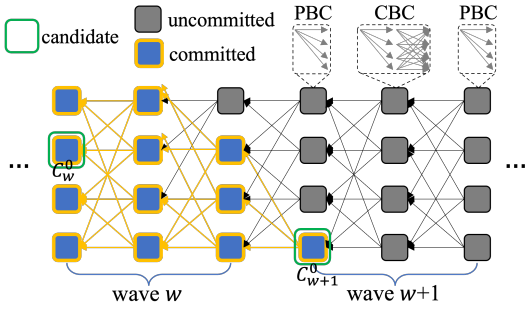


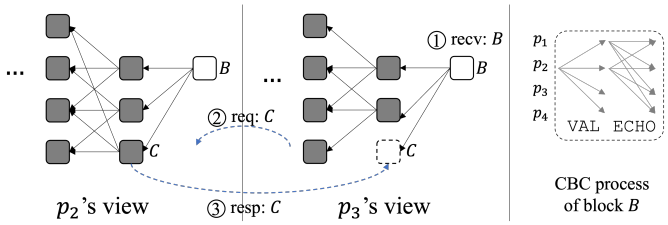
Fig. 3: Structure of LightDAG2

C. Overview of LightDAG1

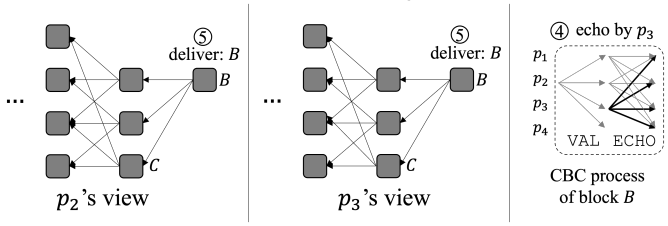
Our first variant of LightDAG, namely LightDAG1, is a simple modification to existing DAG-based protocols that replaces RBC with CBC. By utilizing CBC, LightDAG1 benefits from the reduced number of communication steps required. Due to the consistency property of CBC, each slot can contain at most one block. Therefore, a block can be directly represented by its corresponding slot, denoted as $B_{\langle r,i\rangle}$ or $B_{\langle\langle w,e\rangle,i\rangle}$. Additionally, each wave can have at most one leader block. Thus, we can assume that a leader block is randomly selected through GPC at the end of wave w, which is denoted as L_w.

As shown in Fig. 2, a wave in LightDAG1 consists of three CBC rounds. The leader block of a wave is revealed based on the messages exchanged in the **third** round. Besides, to further reduce latency without compromising safety and liveness, we merge the third round of a wave with the first round of the next wave. Consequently, a round can be represented as $\langle w, e\rangle$, where $e \in \{1, 2, 3\}$, and $\langle w, 3\rangle = \langle w + 1, 1\rangle$. Furthermore, the one-dimensional round number r is given by $2w + e$. If the leader block is referenced by $f + 1$ or more blocks in the **second** round of the wave, it, along with its ancestor blocks, can be committed. This allows LightDAG1 to achieve a best latency of 6 communication steps, or even 5 steps if we count only the first step in the third CBC round.

However, as we mentioned in Section III-B, **CBC blocks cannot guarantee totality**, meaning that blocks broadcast through CBC may only be delivered by a subset of non-faulty replicas. To address this issue, we propose a block retrieval mechanism that helps replicas retrieve missing blocks from each other. A replica participates in the CBC instance of a block B only after it has delivered all the ancestor blocks of B through the block retrieval mechanism.



(a) Retrieve the lacking block



(b) Participate in the CBC process

Fig. 4: An example to show the block retrieval mechanism

D. Overview of LightDAG2

To further reduce latency, we propose LightDAG2 as another variant, where a wave consists of two rounds of *Plain Broadcast* (PBC) and one round of CBC. By PBC, we mean a broadcasting process where the broadcaster directly sends its block B to others, and each receiver delivers B upon receiving it. Compared to RBC and CBC, PBC is more lightweight. As illustrated in Fig. 3, three rounds of a wave are implemented with PBC, CBC, and PBC, respectively. Since **a Byzantine replica can equivocate through PBC**, there may be multiple blocks in a slot, and a block cannot be directly represented by its corresponding slot. Instead, each block in a slot is denoted as $B^j_{\langle r,i\rangle}(j \geq 0)$, where j represents the index of block in the same slot. A slot in the first round will be selected as the leader slot based on GPC messages exchanged in the third round. The blocks in the leader slot are referred to as candidate leader blocks or candidate blocks, denoted as $C^j_w(j \geq 0)$.

If a candidate block is referenced by at least $n - f$ blocks in the third round, the candidate block, along with its ancestor blocks, can be committed. To ensure totality, the block query mechanism is also employed in LightDAG2. In a favorable situation, LightDAG2 achieves a best latency of 4 communication steps, comprising two PBC rounds and one CBC round.

We will explain in Section V that the safety and liveness of LightDAG2 are not compromised by the equivocation in PBC. Intuitively, in a system with $n \geq 3f + 1$ replicas and up to f Byzantine ones, at most one candidate block can be referenced by $n - f$ blocks, ensuring safety. Regarding liveness, although Byzantine replicas can equivocate through PBC to prevent committing within a wave, their equivocation exposes their Byzantine identities and they are excluded in the subsequent waves. Therefore, at most t equivocation attacks can be launched, where t represents the actual number of Byzantine replicas. After that, a candidate block can definitely get committed.

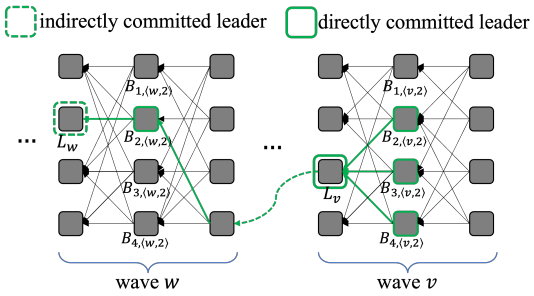


Fig. 5: Indirectly and directly committed leaders

IV. LightDAG1 Design

To address the limitation of CBC which cannot guarantee the totality property, we propose a block retrieval mechanism in Section IV-A. Additionally, as mentioned earlier in Section III-C, all blocks are committed through leader blocks, either directly as leader blocks or indirectly as ancestors of leader blocks. We have discussed how a leader block is committed in Section III-C and will complete the description of committing other non-leader blocks in Section IV-B.

A. Block retrieval mechanism

To elaborate, when a replica p_i receives a block B through the VAL step of CBC from another replica p_j, p_i checks whether it has already delivered all parent blocks of B. If not, p_i sends a request to retrieve the missing blocks by including their hashes in the request. If p_j is non-faulty, it responds to p_i with all the requested blocks. Upon receiving p_j's response, p_i further checks if it has delivered all parents of the blocks included in the response, continuing to retrieve any missing ones. This block retrieval process continues until p_i has delivered all the ancestors of B. Then, p_i participates in the CBC process of B by echoing B.

Fig. 4 provides an example of the block retrieval mechanism involving four replicas. In Fig. 4a, p_2 delivers block C and proposes block B, which directly references C. When p_3 receives B through the VAL step of CBC (① in Fig. 4a), it realizes that it lacks C whose hash is included in B. p_3 sends a request to p_2 to retrieve C (② in Fig. 4a), and p_2 will respond with C (③ in Fig. 4a) if it is a non-faulty replica. Only after delivering C, p_3 participates in the CBC process of B by echoing B (④ in Fig. 4b).). Each replica can then deliver B after receiving $n - f$ ECHO messages (⑤ in Fig. 4b).

B. Block commitment mechanism

The block commitment mechanism in LightDAG1 is similar to existing DAG-based protocols. It is important to note that a leader block L_w of wave w may not be committed in the same wave w if it is not referenced by $f+1$ blocks in round $\langle w, 2\rangle$. However, L_w may be referenced by a later leader block (L_v), while L_v gets referenced by $f+1$ blocks in round $\langle v, 2\rangle$ and gets committed in wave v. Fig. 5 illustrates this scenario with 4 replicas and $f = 1$. On one hand, leader L_w is referenced by only 1 ($< f+1$) block in round $\langle w, 2\rangle$, which cannot be committed in wave w. On the other hand, L_v is referenced by 3 ($\geq f+1$) blocks in round $\langle v, 2\rangle$, allowing it to be committed in wave v. Additionally, L_w is indirectly referenced by L_v, thus getting committed through the commitment of L_v. The commitment of a leader block in its respective wave is referred to as *direct leader commitment* (e.g., the commitment of L_v), while the commitment of a leader block through a later leader block is referred to as *indirect leader commitment* (e.g., the commitment of L_w).

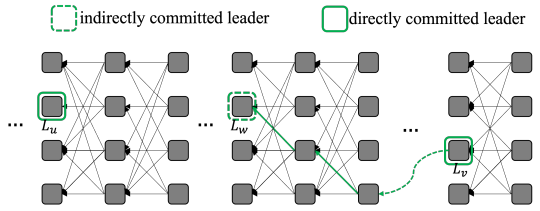


Fig. 6: Search of indirectly committed blocks

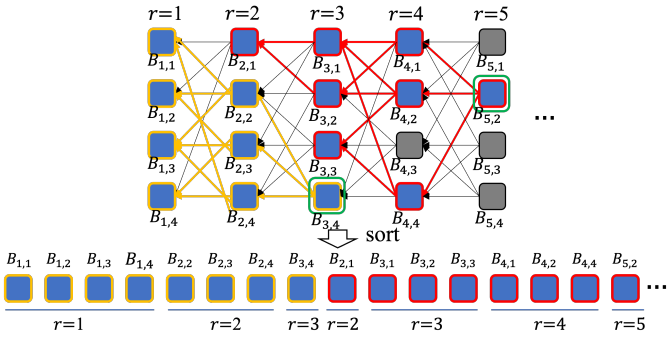


Fig. 7: Sorting blocks in LightDAG1

Once a replica p_i determines that a leader block can be committed directly (indicated by L_v in Fig. 6), meaning it is referenced by $f+1$ blocks in the second round, p_i follows a specific procedure to commit ancestors. Firstly, p_i searches for the most recent directly committed leader in its local storage/ledger (exemplified by L_u in Fig. 6). Next, p_i identifies all leader blocks referenced by L_v and appearing after L_u, such as L_w in Fig. 6. These blocks are indirectly committed leaders through the commitment of L_v and, along with L_v, are denoted as a set $\mathcal{I}_{L_v}$. Subsequently, p_i iterates over the leader blocks in $\mathcal{I}_{L_v}$, starting from the smaller wave number to the larger one. For each L_k in $\mathcal{I}_{L_v}$, p_i sorts all the ancestor blocks of L_k that have not yet been committed. The sorting process is performed first by the block's round number and then by the replica index of the block's proposer. The sorted blocks are considered committed. All the committed blocks are sorted in total order, first by the index of the corresponding committed leader block k, then by the round number of blocks r, and finally by the replica index of block proposers i. Fig. 7 provides an example illustrating the sorting the committed blocks through two leader blocks $B_{3,4}$ and $B_{5,2}$. We describe the details of the block commitment and sorting mechanisms by pseudocode, which are deferred to Section A of Appendix for space limitation.

C. Correctness analysis

The correctness analysis of LightDAG1 encompasses two main aspects: safety and liveness.

1) Safety analysis: In the block commitment mechanism described in Section IV-B, all blocks in LightDAG1 are committed through leader blocks. These leader blocks, whether directly or indirectly committed, can be sorted in a total order and assigned incremental indices k. Based on the consistency property of CBC, we only need to prove that the committed leaders with the same index are identical across different replicas. This safety property can be expressed through Theorem 2, the proof of which relies on Lemma 1. Due to space limitations, the proof of Lemma 1 is deferred to Section B of Appendix.

LEMMA 1. *In LightDAG1, if two leader blocks (L and L') are directly committed by two non-faulty replicas, respectively, then either $L \in \mathcal{A}_{L'}$ or $L' \in \mathcal{A}_L$.*

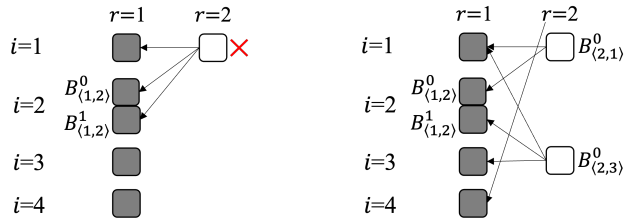
THEOREM 2 (Safety of LightDAG1). *If two leader blocks (L and L') are committed by two non-faulty replicas with the same index, respectively, then $L = L'$.*

Proof. Let D and D' represent directly committed leaders corresponding to L and L', respectively. If L (respectively, L') is directly committed, then $D = L$ (respectively, $D' = L'$). According to Lemma 1, either $D \in \mathcal{A}_{D'}$ or $D' \in \mathcal{A}_D$. Without loss of generality, let us assume $D \in \mathcal{A}_{D'}$. Since $L \in \mathcal{A}_D$, we have $L \in \mathcal{A}_{D'}$ and $\mathcal{A}_L \prec \mathcal{A}_{D'}$. On the other hand, $\mathcal{A}_{L'} \prec \mathcal{A}_{D'}$. In other words, both sequences $\mathcal{A}_L$ and $\mathcal{A}_{L'}$ are prefixes of the same sequence $\mathcal{A}_{D'}$. Therefore, either $\mathcal{A}_L \prec \mathcal{A}_{L'}$ or $\mathcal{A}_{L'} \prec \mathcal{A}_L$. Since L and L' are assigned the same index, L must be identical to L', thus concluding the proof of safety. □

2) Liveness analysis: For brevity, we refer to blocks proposed by non-faulty replicas as correct blocks. Since a client sends its transaction to every replica, as long as the probability of at least one correct block being directly committed in a wave is not infinitely small, this transaction will eventually get committed, ensuring liveness. Additionally, as a leader block must reference at least $n - f$ blocks, with at least $f + 1$ of them are correct, we interpret the liveness property through Theorem 3.

THEOREM 3 (Liveness of LightDAG1). *The probability that a leader block gets directly committed in a wave is not infinitely small.*

Proof. If a block B in round $\langle w, 1\rangle$ is referenced by at least $f + 1$ blocks in round $\langle w, 2\rangle$, it will be indirectly referenced by every block in round $\langle w, 3\rangle$, as each block in $\langle w, 3\rangle$ must reference at least $n - f$ blocks in $\langle w, 2\rangle$. If B is selected as the leader block of wave w, it can be committed directly. Our proof involves determining the number of blocks in round $\langle w, 1\rangle$ that are referenced by at least $f + 1$ blocks in $\langle w, 2\rangle$, forming a set. Subsequently, we calculate the probability of the leader block coincidentally belonging to this set, indicating the likelihood of directly committing the leader block.



(a) One block in each slot can be referenced by a block

(b) Multiple blocks in a slot may be referenced by different blocks

Fig. 8: Examples to show block references to a slot. A block directly referencing two contradictory blocks within a slot is deemed unallowable, as indicated by the cross sign in (a). In contrast, distinct blocks are permitted to reference contradictory blocks within a slot, as illustrated in (b).

According to the threshold property of GPC, when the leader block of a wave w is revealed, at least one correct block in round $\langle w, 3\rangle$ is proposed. At this point, at least $n-f$ blocks in round $\langle w, 2\rangle$ have completed the CBC process, denoted as $\mathcal{T}$. Since each block in $\mathcal{T}$ directly references at least $n - f$ blocks in round $\langle w, 1\rangle$, the blocks in round $\langle w, 1\rangle$ will be directly referenced $(n - f) \cdot (n - f)$ times in total. In other words, there are at least $(n - f) \cdot (n - f)$ references from round $\langle w, 2\rangle$ to $\langle w, 1\rangle$.

To facilitate presentation, we say a block B is assigned a reference if B is directly referenced by a later block. To determine the minimum number of blocks in round $\langle w, 1\rangle$ that are referenced at least $f + 1$ times, we attempt to distribute the references by initially assigning f references to each block in round $\langle w, 1\rangle$, taking up $n \cot f$ references. Each block in round $\langle w, 1\rangle$ can then be assigned at most $n - 2f$ references. Since $(n-f)\cdot(n-f) \geq (n-f)\cdot(2f+1) \geq n\cdot f+(n-f)\cdot(f+1)-f\cdot(f+1) = n\cdot f+(f+1)\cdot(n-2f)$, more than $f+1$ blocks in round $\langle w, 1\rangle$ will be assigned extra references in addition to the initial f references.

In other words, there are at least $f + 2$ blocks in round $\langle w, 1\rangle$, denoted as a set $\mathcal{G}$, each being directly referenced at least $f+1$ times. If the leader block L_w is a member of $\mathcal{G}$, it can be directly committed. The probability of $L_w \in \mathcal{G}$ is larger than 1/3, which is not infinitely small, thus proving Theorem 3 and liveness. □

V. LIGHTDAG2 DESIGN

In this section, we address the issue of Byzantine equivocations due to PBC utilized in the first and third rounds of a wave. To aid understanding, we present LightDAG2's design in a step-by-step manner, with each step as a newly added rule. Additionally, we briefly describe the block retrieval and commitment mechanisms in LightDAG2, which closely resemble those in LightDAG1.

A. Block creation and broadcast rules

1) Rule 1: First, we introduce Rule 1, which applies to blocks in all rounds. Under Rule 1, a block is allowed to directly reference at most one block in each slot of the previous

round. This prevents a single block from directly referencing two contradictory blocks equivocated by Byzantine replicas, as illustrated in Fig. 8a.

Rule 1

A block must directly reference at least $n - f$ blocks from the previous round, with each of these referenced blocks occupying a distinct slot.

Rule 2

- If a replica p_x has voted for a block B that directly references a block $C^0_{\langle r,i\rangle}$ in the previous PBC round, it will refrain from voting for another block D that directly references $C^1_{\langle r,i\rangle}$.
- Instead, p_x will send $C^0_{\langle r,i\rangle}$ to replica p_y who proposes D.
- Upon receiving $C^0_{\langle r,i\rangle}$, p_y becomes aware of the Byzantine identity of p_i who is $C^0_{\langle r,i\rangle}$'s proposer. p_y will then repropose a new block D' that references none of p_i's blocks. Additionally, D' will include both $C^0_{\langle r,i\rangle}$ and $C^1_{\langle r,i\rangle}$ as a proof of p_i's Byzantine identity.
- Subsequently, p_x can vote for D'.

2) Rule 2: However, there may be multiple contradictory blocks in a slot being referenced by different blocks, as shown in Fig. 8b. To address this, we introduce Rule 2, specifically for the CBC round in each wave. In the CBC process, which consists of the VAL step and ECHO step, a replica p is said to vote for a block B if it broadcasts ECHO messages for B. Rule 2 aims to resolve conflicts by preventing replicas from voting for contradictory blocks. Fig. 9 illustrates an example of Rule 2. From the perspective of replica p_x, which has already voted for block B, it will refrain from voting for block D (as shown in Fig. 9a). Instead, p_x will send the contradictory block to the proposer of D, replica p_y, as depicted in Fig. 9b. p_y will then propose a new block D' that includes both blocks C^0 and C^1 as a proof, and broadcast D' using CBC. Subsequently, p_x can vote for D', as shown in Fig. 9c.

In essence, Rule 2 ensures that, for any two blocks that are delivered in the same CBC round, they will not directly reference two contradictory blocks from the previous round. In other words, situations like the one depicted in Fig. 10a become impossible under Rule 2. However, it is still possible for multiple blocks in the same CBC slot to be delivered, as illustrated in Fig. 10b. This scenario can occur even if the proposer of B or B' is a non-faulty replica, presenting a challenge to liveness since there may be more than n blocks delivered in the CBC round.

Rule 3

- Once a replica proposes or votes for a CBC block in wave w, it will abstain from voting for any CBC block in a previous wave $v (v < w)$.

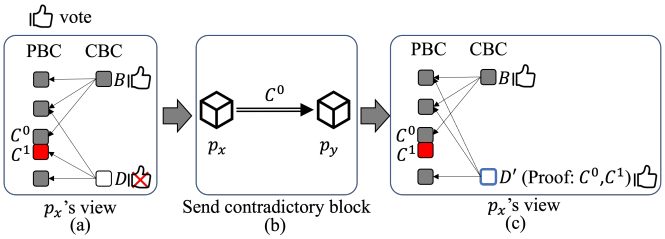


Fig. 9: A example to describe Rule 2

(a) Impossible situation (b) Possible situation

Fig. 10: Delivery situations in a CBC round

- Upon receiving a Byzantine proof for a replica p_b, a replica will refrain from referencing any block proposed by p_b. Additionally, it will include the Byzantine proof in its subsequent proposed block.
- Upon receiving a Byzantine proof for a replica p_b, a replica will abstain from voting for any CBC block B that directly references a block proposed by p_b. Furthermore, it will send the Byzantine proof to B's proposer.

3) Rule 3: To address the liveness challenge illustrated in Fig. 10b, LightDAG2 introduces Rule 3 to the CBC round. For brevity, the proof of a replica's Byzantine identity is referred to as Byzantine proof. As demonstrated by Lemma 8 in Section V-C, under Rule 3, if multiple blocks are delivered in a CBC slot, some Byzantine replicas will be recognized and excluded in subsequent waves, which contributes to liveness.

Rule 4

- A replica will first search for the non-empty leader slot with the highest wave number, denoted as s_w, where the wave number is w.
- Based on the blocks in round $\langle w, 3\rangle$, the replica will determine a unique block for s_w. Since s_w is non-empty, at least one block in round $\langle w, 3\rangle$ does not reference $\perp$ in s_w. Additionally, all blocks in round $\langle w, 3\rangle$ reference at most one block in s_w. Therefore, from the replica's view, there is one and only one block in s_w, denoted as B_{s_w}.
- After excluding all the slots that have already been determined by B_{s_w}, the replica proceeds to determine a unique block for each remaining slot. If a slot contains only one block, the replicas simply determines that block for the slot. Otherwise, the replica determines one randomly and contains the determination in the newly proposed block.

4) Rule 4: Similar to LightDAG1, all blocks in LightDAG2 are committed through the candidate blocks in the leader slots. However, since some blocks are broadcast through PBC, a candidate block may reference contradictory blocks, which poses a safety challenge. To tackle this challenge, Rule 4 is proposed specifically for the blocks in the first rounds. Under Rule 4, a block in the first round must determine a unique block for each slot in its ancestor view, particularly in cases where multiple blocks exist in a slot. As demonstrated by Lemma 4 in Section V-C, in a wave w, all blocks in round $\langle w, 3\rangle$ will not reference contradictory blocks in a slot in round $\langle w, 1\rangle$. In other words, for a specific slot $\langle\langle w, 1\rangle, i\rangle$ in the first round, every block in round $\langle w, 3\rangle$ will either indirectly reference the same block $B_{\langle\langle w,1\rangle,i\rangle}$ or not reference any block in that slot. We refer to a block that does not reference any block in a slot as referencing $\perp$ in that slot.

Fig. 11 provides an example to illustrate Rule 4. When a replica p wants to propose a new block B in the first round of wave v, it will search for the non-empty leader slot with the maximum wave number. In Fig. 11, the leader slot in wave $w+1$ will be skipped since it is empty, and the leader slot in wave w is chosen. Following Rule 2, there will be only one block in the leader slot of wave w, denoted as C in the example. As a block in the first round of wave w, C must have determined all the blocks in its ancestor view, indicated by the blue shadow. Replica p will determine all blocks in its ancestor view that have not been determined by C. For slots with more than one block, such as slot $\langle\langle w, 1\rangle, 3\rangle$ and $\langle\langle w+1, 2\rangle, 1\rangle$, p can randomly determine a block in the slot and incorporate its determination into the new block. It is important to note that either through its own determination or through another candidate block in its ancestor view (e.g., C in Fig. 11), a candidate block can uniquely determine a block for each slot.

B. Block retrieval & commitment mechanisms

In LightDAG2, the block retrieval mechanism remains consistent with LightDAG1. A replica will only deliver a PBC block B or vote for a CBC block B if all of B's ancestor blocks have been delivered. In the case of a candidate block in the leader slot, if it is referenced by $n-f$ or more blocks in the last round of the wave, this candidate block can be committed. Once committed, the candidate block will proceed to sort and commit all the blocks that have not yet been committed by the previously committed candidate block. The remaining sorting and commitment mechanisms in LightDAG2 are the same as those in LightDAG1.

C. Correctness analysis

1) Safety analysis: In LightDAG2, all committed candidate blocks can be sorted based on their wave numbers and assigned incremental indices k. We define $\mathcal{U}$ as a block sequence representing a candidate block's ancestors, which includes the unique block for each slot and is sorted first by the round number and then by the replica index. As all blocks in LightDAG2 are committed through the commitment of candidate blocks, we interpret safety property as Theorem 6,

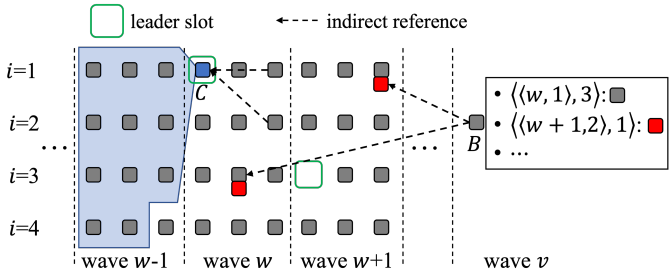


Fig. 11: A example to describe Rule 4

whose proof relies on Lemma 4 and Lemma 5. We also defer the proof of lemmas to Section B of Appendix.

LEMMA 4. *In a wave of LightDAG2, all blocks in the third round will not reference contradictory blocks in the first round.*

LEMMA 5. *In LightDAG2, if two candidate blocks (C and C') are directly committed by two non-faulty replicas, respectively, then either $C \in \mathcal{U}_{C'}$ or $C' \in \mathcal{U}_C$.*

THEOREM 6 (Safety of LightDAG2). *If two candidate blocks (C and C') are committed by two non-faulty replicas with the same index, respectively, then $C = C'$.*

Proof. The proof of Theorem 6 follows a similar approach to the proof of Theorem 3. Let D and D' denote the directly committed candidates corresponding to C and C'. According to Lemma 5, we have $D \in \mathcal{U}_{D'}$ or $D' \in \mathcal{U}_D$, and we assume $D \in \mathcal{U}_{D'}$ without loss of generality. Since C is referenced by D, it implies that $C \in \mathcal{U}_{D'}$, and consequently, $\mathcal{U}_C \prec \mathcal{U}_{D'}$. On the other hand, $\mathcal{U}_{C'} \prec \mathcal{U}_{D'}$, meaning that both $\mathcal{U}_C$ and $\mathcal{U}_{C'}$ are prefixes of $\mathcal{U}_{D'}$. Therefore, we can conclude that either $\mathcal{U}_C \prec \mathcal{U}_{C'}$ or $\mathcal{U}_{C'} \prec \mathcal{U}_C$. Since C and C' are committed with the same index, in either case of $\mathcal{U}_C \prec \mathcal{U}_{C'}$ and $\mathcal{U}_{C'} \prec \mathcal{U}_C$, we must have $C = C'$. □

2) Liveness analysis: We define a block as a reproposed block if it is reproposed by a replica p after p receives a Byzantine proof. Note that all reproposed blocks are CBC blocks. According to Rule 2 of LightDAG2, a reproposed block will include the Byzantine proof. As a result, any replica that receives the reproposed block can recognize the Byzantine identity of the corresponding replica. The liveness of LightDAG2 is interpreted through Theorem 10, which asserts that the expected number of waves required to directly commit a candidate block is less than $3(t+1)$, where t represents the actual number of Byzantine replicas. We prove Theorem 10 based on three lemmas, whose proof is presented in Appendix due to space limitation.

LEMMA 7. *In LightDAG2, if a reproposed block with a Byzantine proof for p_b is delivered in wave w, any CBC block of wave $v (v > w)$ that directly references a block proposed by p_b will not be delivered.*

LEMMA 8. *In LightDAG2, if a reproposed block with a Byzantine proof for p_b is delivered in wave w, each replica will*

recognize p_b's Byzantine identity in or before round $\langle w+1, 3\rangle$.

LEMMA 9. *In a wave w of LightDAG2, if no reproposed block is delivered, the probability that a candidate block gets directly committed is larger than $\frac{1}{3}$.*

THEOREM 10 (Liveness of LightDAG2). *In a system with $t(t \leq f)$ Byzantine replicas, the probability that a candidate block gets directly committed within $t+1$ waves exceeds 1/3.*

Proof. Denote a system with $k(k \leq t)$ unrecognized Byzantine replicas as $\Im_k$ and denote the probability that a candidate block gets directly committed in $\Im_k$ as $g(k)$. It is easy to know $g(0) > \frac{2}{3}$. We will consider two cases in a wave w, based on whether a reproposed block is delivered:

- **Case 1:** No reproposed block is delivered.
- **Case 2:** At least one reproposed block is delivered.

Let ρ_t be the probability that Case 1 occurs in a system $\Im_k$, and q_k be the probability that a leader gets directly committed in Case 1. According to Lemma 9, we have $q_k > \frac{1}{3}$. If Case 2 occurs in a wave, it implies that one or more Byzantine replica will be recognized. We denote the set of recognized Byzantine replicas as $\mathcal{S}$, and let $s = |\mathcal{S}|\,(s \geq 1)$. According to Lemma 7, any CBC block in wave $v(v > w)$ that references a block proposed by any replica in $\mathcal{S}$ will not be delivered. According to Lemma 8, each replica in or after the round $\langle w+1, 3\rangle$ will recognize the Byzantine identities of replicas in $\mathcal{S}$. Therefore, if Case 2 occurs in wave w, $\mathcal{S}$ will be excluded from the consensus protocol after wave w, and the number of unrecognized replicas in the system will be $k-s$. Hence, we can express $g(k)$ as $g(k) = \rho_k \cdot q_k + (1-\rho_k) \cdot g(k-s)$. Next, we will prove $g(k) > \frac{1}{3}$ by mathematical induction.

- **Base case:** $g(0) > \frac{2}{3} > \frac{1}{3}$.
- **Inductive step:** For a given number m, we assume that $\forall i \leq m : g(i) > \frac{1}{3}$. According to Equation 1, we have $g(m+1) > \frac{1}{3}$.

$$\begin{aligned} g(m+1) &= \rho_{m+1} \cdot q_{m+1} + (1-\rho_{m+1}) \cdot g(m+1-s) \\ &> \rho_{m+1} \cdot \frac{1}{3} + (1-\rho_{m+1}) \cdot \frac{1}{3} = \frac{1}{3} \end{aligned} \tag{1}$$

Therefore, by mathematical induction, we have $g(k) > \frac{1}{3}$ for all $k \leq t$. In other words, for each $k \leq t$, the probability that a candidate block is directly committed within $k+1$ waves is larger than $\frac{1}{3}$. This implies that the expected number of waves required to directly commit a candidate block in $\Im_t$ is smaller than $3(t+1)$. □

VI. IMPLEMENTATION & EVALUATION

To evaluate LightDAG, we implement prototype systems for both variants and compare them to the state-of-the-art, namely Tusk [10] and BullShark [9].

A. Implementation & settings

We implement all of LightDAG, Tusk, and BullShark in Golang using a common framework to ensure a fair and consistent comparison. The total lines of code for these implementations amount to approximately 4,600. We utilize various open-source libraries, such as kyber[1] for threshold signature schemes and go-msgpack[2] for encoding and decoding functions. The CBC protocol implementation is based on Dolev's paper [14]. Regarding the RBC protocol in Tusk and BullShark, it is implemented based on Cachin's paper [24].

All experiments are conducted on Alibaba Cloud, with each replica deployed as an ECS.g6e.xlarge instance, featuring 4 vCPU and 16 GB memory. These replicas are deployed on four continents to mimic a distributed setting and are connected through peer-to-peer network links with a bandwidth of 100 Mbps. The transaction size is consistently set to 128 bytes, and each group of experiments is repeated five times to reduce experimental errors. The focus of the experiments is on two metrics: latency and throughput. Latency is measured as the time taken by a transaction to be committed from the moment it is proposed, while throughput is computed as the number of committed *Transactions Per Second* (TPS).

We examine both favorable situations, where all replicas adhere to the protocol honestly, and unfavorable situations, where the adversary orchestrates Byzantine replicas to act maliciously. Given that the adversary is incapable of compromising the safety or liveness properties, as demonstrated by the correctness proofs of Tusk, BullShark, and this work, its impact is limited to damaging protocol efficiency. To achieve this, we consider the most potent attack that the adversary can mount against each protocol. As Tusk and LightDAG1 leverages a broadcast protocol that ensures consistency without introducing optimistic paths, the adversary's strategy involves crashing Byzantine replicas to reduce the number of proposed blocks in each round. BullShark, on the other hand, can be targeted by delaying blocks from leaders to disrupt the optimistic path. Regarding LightDAG2, the adversary schedules one Byzantine replica each time, to broadcast contradictory blocks in the first round of a wave, enticing each replica to repropose blocks in the second round. This results in more than n blocks being generated in the second round. When the leader block is revealed and assumed to be referenced by some (possibly more than $f+1$) blocks in the second round, denoted as $\mathcal{R}$, the adversary strategically schedules block delivery to prevent blocks in the third round from directly referencing blocks in $\mathcal{R}$.

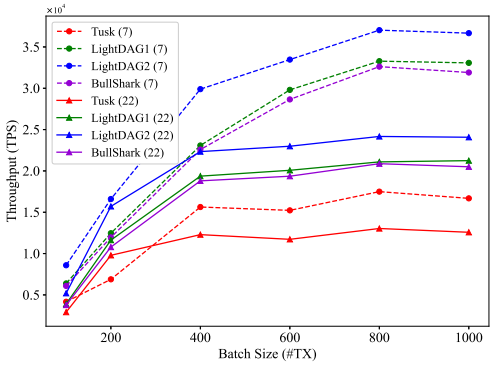
B. Basic performance under favorable situations

We compare the performance of LightDAG, Tusk, and BullShark in two settings: one with 7 replicas and the other with 22 replicas. We increase the batch size, representing the number of transactions contained in a block, from 100 to 1,000. The experimental results are depicted in Fig. 12. By analyzing the performance within the same setting, we observe that both LightDAG1 and LightDAG2 can consistently outperform Tusk or BullShark. Moreover, LightDAG2 exhibits better performance than LightDAG1 in most cases.

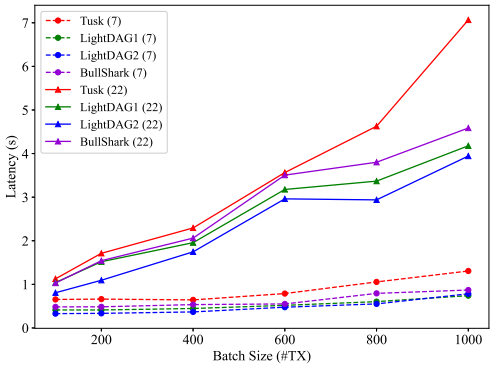
When the system comprises 22 replicas and the batch size is set as 1,000, LightDAG1 and LightDAG2 deliver 1.69x and

[1]https://github.com/dedis/kyber

[2]https://github.com/hashicorp/go-msgpack

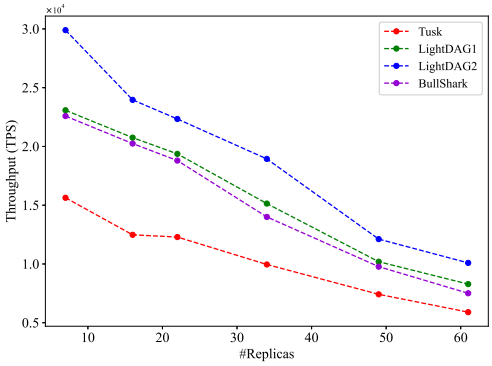


(a) Throughput comparison

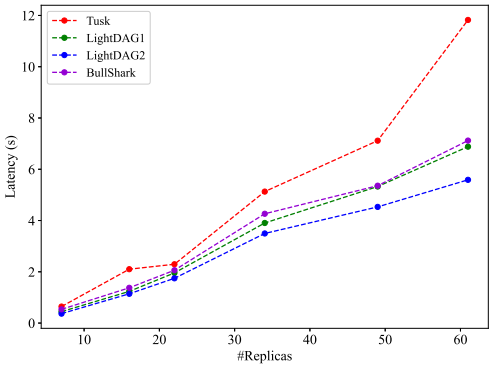


(b) Latency comparison

Fig. 12: Comparison by increasing the batch size



(a) Throughput comparison



(b) Latency comparison

Fig. 13: Comparison by increasing the number of replicas

1.91x higher throughput than Tusk, respectively. In terms of latency, LightDAG1 and LightDAG2 reduce Tusk's latency by 41% and 45%, respectively. In contrast to BullShark, LightDAG1 demonstrates a modest performance enhancement under these favorable situations. Conversely, LightDAG2 achieves a remarkable 17.4% improvement in throughput and a 14.0% reduction in latency within the context of 22 replicas and a batch size of 1000.

C. Scalability under favorable situations

The scalability of a protocol is measured by increasing the number of replicas. In our experiments, we keep the batch size fixed at 400 and gradually increase the number of replicas from 7 to 61. The results are presented in Fig. 13. Despite the performance degradation as the number of replicas increases, both LightDAG1 and LightDAG2 consistently outperform Tusk or BullShark. Additionally, as depicted in Fig. 13b, the slope of the lines representing LightDAG1 and LightDAG2 is smaller than that of Tusk. This indicates that the increase in latency for LightDAG is slower, highlighting its superior scalability. An interesting phenomenon is observed in Fig. 13a where all curves converge as the replica count increases. This convergence is attributed to the escalating communication overhead that accompanies the growth in the number of replicas, subsequently exerting a negative impact on system throughput. This phenomenon causes all curves to trend toward lower and lower values, towards an ultimate and same value of 0, which accounts for their convergence.

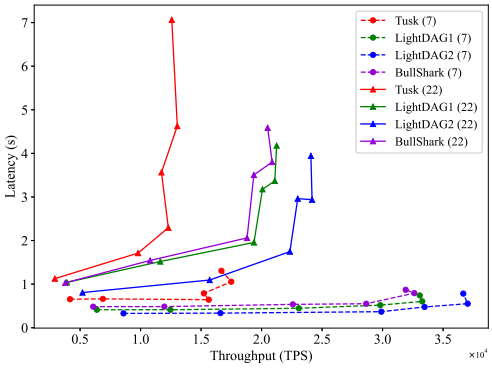


Fig. 14: Trade-off between latency and throughput under favorable situations

D. Latency v.s. throughput under favorable situations

As already illustrated in Fig. 12, the throughput initially increases and then stabilizes at a peak value as the batch size increases, while the latency continues to increase. This phenomenon occurs because the system becomes saturated when the batch size reaches a certain point, and further increasing it only results in higher latency without improving the throughput. This trade-off between latency and throughput can be visualized by plotting both metrics in a figure.

We also conduct experiments in two settings that contain different numbers of replicas, gradually increasing the batch size until the peak throughput is reached. The results are presented in Fig. 14. It is observed that both LightDAG1

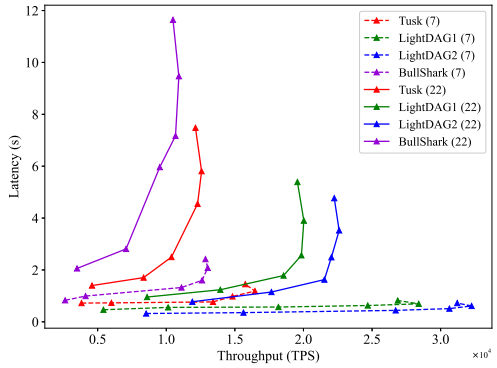


Fig. 15: Trade-off between latency and throughput under unfavorable situations

and LightDAG2 achieve higher peak throughputs than Tusk or BullShark, with LightDAG2 consistently achieving the highest throughput. For instance, in a system with 22 replicas, Tusk and BullShark achieve a peak throughput of 13.0k TPS and 20.5k TPS, while LightDAG1 and LightDAG2 achieve 21.2k TPS and 24.1k TPS, respectively.

E. Latency v.s. throughput under unfavorable situations

We also examine the trade-off between latency and throughput under unfavorable situations, illustrating latency in unsaturated conditions and peak throughput in saturated conditions. Experimental results are presented in Fig. 15. Notably, BullShark delivers the poorest performance, attributed to the failure of its optimistic path and the prolonged switch from the optimistic path to the pessimistic path. LightDAG1 consistently outperforms Tusk, aligning with the results analyzed in Table I.

An intriguing observation is that LightDAG demonstrates the best performance, seemingly contradicting the findings in Table I. This discrepancy is clarified by the fact that the result $12(t+1)$ pertains to the worst-case scenario where the adversary successfully prevents block commitment in continuous t waves, a condition challenging to achieve in practical settings. Moreover, each time the adversary successfully executes an attack, one of the Byzantine replicas is identified and excluded, resulting in improved performance in subsequent waves.

VII. Related Work

A. Non-DAG protocols

The study of BFT consensus dates back to the 1980s, with the Byzantine general problem being the pioneering work [25], [26]. Initially, BFT consensus studies are conducted without adopting a specific topology, where requests at different positions of the ledger are agreed upon independently. The well-known representative of this approach is PBFT [6]. Subsequent to PBFT, various works aim to reduce latency by integrating a fast path, either through the use of trusted hardware [27], [28], relaxing the resilience [29], [30], or assuming a flexible fault model [31], [32]. However, the independent-position scheme makes these protocol pretty complex and hard to understand or implement.

The emergence of Bitcoin [33] and Ethereum [34] introduces a new perspective to consensus design by organizing multiple transactions into blocks and chaining all blocks together [35], [36]. This chain-based topology simplifies the consensus protocol and improves system performance. Consequently, many modern BFT works start utilizing the chain-based topology, such as Tendermint [37], Pili [38], Hot-Stuff [7], and Streamlet [39]. However, this chain-based topology can only process blocks in a sequential manner, which limits system throughput. Another line of works delves into enhancing consensus protocols by incorporating accountability properties [3] or voting validity properties [2]. Alternatively, some investigations aim to refine these protocols by mitigating assumptions, such as eliminating the consideration of failure counts [4].

B. DAG-based protocols

To parallel block processing, the DAG topology is introduced in consensus design [40], with DAGRider being a significant milestone attempt [8]. Following DAGRider, a series of works aim to reduce the rounds in a wave to decrease latency, whose representatives include Tusk [10] and BullShark [9]. However, even though these protocols can reduce the number of rounds in a wave to smaller, each round relies on the heavy broadcasting protocol (i.e., RBC), which consists of at least three communication steps. Therefore, all these DAG-based protocols still suffer from high latency.

Another line of related works is the public DAG-based blockchain, which combines the DAG topology with a public blockchain system. Examples of such works include Conflux [41], OHIE [42], and Occam [43]. Although these works also need to achieve the Byzantine consensus on data processing, they are targeted at a different model compared to our work. Specifically, they operate under a permissionless model, while our work assume a permissioned model. Additionally, consensus in a permissioned model typically provides much better performance than in a permissionless model.

VIII. Conclusion

Existing DAG-based protocols suffer from high latency due to the use of the heavy broadcasting protocol (i.e., RBC). To deal with this problem, we propose LightDAG, which replaces RBC with lightweight broadcasting protocols like CBC and PBC. We specifically present two variants of LightDAG, namely LightDAG1 and LightDAG2, that offer a trade-off between best latency and expected worst latency. Comprehensive experiments are conducted to evaluate LightDAG, which demonstrates its good performance.

Acknowledgment

This work was supported by National Key Research and Development Program of China under Grant No. 2021YFB2700700, Key Research and Development Program of Hubei Province No. 2021BEA164. Jiang Xiao is the corresponding author of the paper.

REFERENCES

[1] H. Jin and J. Xiao, "Towards trustworthy blockchain systems in the era of "internet of value": development, challenges, and future trends," *Science China Information Sciences*, vol. 65, pp. 1–11, 2022.

[2] Z. Xu, Y. Li, C. Feng, and L. Zhang, "Exact fault-tolerant consensus with voting validity," in *Proceedings of the 2023 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2023, pp. 842–852.

[3] P. Civit, S. Gilbert, V. Gramoli, R. Guerraoui, and J. Komatovic, "As easy as abc: Optimal (a) ccountable (b) yzantine (c) onsensus is easy!" in *Proceedings of the 2022 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2022, pp. 560–570.

[4] P. Khanchandani and R. Wattenhofer, "Byzantine agreement with unknown participants and failures," in *Proceedings of the 2021 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2021, pp. 952–961.

[5] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *Proceedings of the 2017 IEEE International Congress on Big Data*. IEEE, 2017, pp. 557–564.

[6] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *Proceedings of the 1999 USENIX Symposium on Operating Systems Design and Implementation*. USENIX, 1999, pp. 173–186.

[7] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: BFT consensus with linearity and responsiveness," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. ACM, 2019, pp. 347–356.

[8] I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman, "All you need is DAG," in *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*. ACM, 2021, pp. 165–175.

[9] A. Spiegelman, N. Giridharan, A. Sonnino, and L. Kokoris-Kogias, "Bullshark: DAG BFT protocols made practical," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2022, pp. 2705–2718.

[10] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Narwhal and tusk: a DAG-based mempool and efficient BFT consensus," in *Proceedings of the 2022 ACM European Conference on Computer Systems*. ACM, 2022, pp. 34–50.

[11] F. M. Benčić and I. P. Žarko, "Distributed ledger technology: Blockchain compared to directed acyclic graph," in *Proceedings of the 2018 IEEE International Conference on Distributed Computing Systems*. IEEE, 2018, pp. 1569–1570.

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

[13] G. Bracha, "Asynchronous Byzantine agreement protocols," *Information and Computation*, vol. 75, no. 2, pp. 130–143, 1987.

[14] D. Dolev, "The Byzantine generals strike again," *Journal of algorithms*, vol. 3, no. 1, pp. 14–30, 1982.

[15] J.-M. Chang and N. F. Maxemchuk, "Reliable broadcast protocols," *ACM Transactions on Computer Systems*, vol. 2, no. 3, pp. 251–273, 1984.

[16] S. Das, Z. Xiang, and L. Ren, "Asynchronous data dissemination and its applications," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2021, pp. 2705–2721.

[17] E. Kokoris Kogias, D. Malkhi, and A. Spiegelman, "Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures." in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2020, pp. 1751–1767.

[18] S. Das, T. Yurek, Z. Xiang, A. Miller, L. Kokoris-Kogias, and L. Ren, "Practical asynchronous distributed key generation," in *Proceedings of the 2022 IEEE Symposium on Security and Privacy*. IEEE, 2022, pp. 2518–2534.

[19] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, "Secure and efficient asynchronous broadcast protocols," in *Proceedings of the 2001 Annual International Cryptology Conference*. Springer, 2001, pp. 524–541.

[20] M. K. Reiter, "Secure agreement protocols: Reliable and atomic group multicast in rampart," in *Proceedings of the 1994 ACM Conference on Computer and Communications Security*. ACM, 1994, pp. 68–80.

[21] V. Shoup, "Practical threshold signatures," in *Proceedings of the 2000 International Conference on the Theory and Application of Cryptographic Techniques*. Springer, 2000, pp. 207–220.

[22] R. Friedman, A. Mostefaoui, and M. Raynal, "Simple and efficient oracle-based consensus protocols for asynchronous Byzantine systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 1, pp. 46–56, 2005.

[23] A. Mostéfaoui, H. Moumen, and M. Raynal, "Signature-free asynchronous Byzantine consensus with $t < n/3$ and $o(n^2)$ messages," in *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*. ACM, 2014, pp. 2–9.

[24] C. Cachin and S. Tessaro, "Asynchronous verifiable information dispersal," in *Proceedings of the 2005 IEEE Symposium on Reliable Distributed Systems*. IEEE, 2005, pp. 191–201.

[25] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *Journal of the ACM*, vol. 27, no. 2, pp. 228–234, 1980.

[26] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, 1982.

[27] G. S. Veronese, M. Correia, A. N. Bessani, L. C. Lung, and P. Verissimo, "Efficient Byzantine fault-tolerance," *IEEE Transactions on Computers*, vol. 62, no. 1, pp. 16–30, 2011.

[28] J. Liu, W. Li, G. O. Karame, and N. Asokan, "Scalable Byzantine consensus via hardware-assisted secret sharing," *IEEE Transactions on Computers*, vol. 68, no. 1, pp. 139–151, 2018.

[29] G. G. Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. Reiter, D.-A. Seredinschi, O. Tamir, and A. Tomescu, "Sbft: a scalable and decentralized trust infrastructure," in *Proceedings of the 2019 IEEE Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2019, pp. 568–580.

[30] J.-P. Martin and L. Alvisi, "Fast Byzantine consensus," *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 3, pp. 202–215, 2006.

[31] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: speculative Byzantine fault tolerance," in *Proceedings of the 2007 ACM SIGOPS Symposium on Operating Systems Principles*. ACM, 2007, pp. 45–58.

[32] X. Dai, L. Huang, J. Xiao, Z. Zhang, X. Xie, and H. Jin, "Trebiz: Byzantine fault tolerance with Byzantine merchants," in *Proceedings of the 2022 Annual Computer Security Applications Conference*. IEEE, 2022, pp. 923–935.

[33] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized business review*, p. 21260, 2008.

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

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

[36] V. Gramoli, "From blockchain consensus back to Byzantine consensus," *Future Generation Computer Systems*, vol. 107, pp. 760–769, 2020.

[37] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," Ph.D. dissertation, University of Guelph, 2016.

[38] T. H. Chan, R. Pass, and E. Shi, "Pili: An extremely simple synchronous blockchain," *Cryptology ePrint Archive*, 2018.

[39] B. Y. Chan and E. Shi, "Streamlet: Textbook streamlined blockchains," in *Proceedings of the 2020 ACM Conference on Advances in Financial Technologies*. ACM, 2020, pp. 1–11.

[40] M. A. Schett and G. Danezis, "Embedding a deterministic BFT protocol in a block DAG," in *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*. ACM, 2021, pp. 177–186.

[41] 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*. IEEE, 2020, pp. 515–528.

[42] 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.

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

Algorithm 1 Block committing and sorting mechanism (in wave v)

Let $sort_{r,p}$ denote the sorting function first by round numbers and then by the replica indices of block creators.

upon the leader block L_v is selected out **do**
  if L_v is referenced by $2f$+1 blocks in round $\langle v, 2\rangle$ **then**
    $u \leftarrow v - 1$
    while L_u is not committed
      $u \leftarrow u - 1$
    $w \leftarrow u + 1$
    while $(w \leq v)$
      if L_w is delivered and is an ancestor of L_v **then**
        $\mathcal{B}_{L_w} \leftarrow$ all uncommitted ancestor blocks of L_w
        $sort_{r,p}(\mathcal{B}_{L_w})$
        commit blocks in $\mathcal{B}_{L_w}$ one by one; **commit** L_w
      $w \leftarrow w + 1$

APPENDIX

A. Pesudocode for block committing and sorting

Pesudecode for block committing and sorting in wave v is described in Algorithm 1.

B. Proof of lemmas

LEMMA 1. *In LightDAG1, if two leader blocks (L and L') are directly committed by two non-faulty replicas, respectively, then either $L \in \mathcal{A}_{L'}$ or $L' \in \mathcal{A}_L$.*

Proof. If L and L' have the same wave number, according to the consistency property of CBC, L must be identical to L'. Since we stipulate that a block is also an ancestor of itself, we have $L \in \mathcal{A}_{L'}$ and $L' \in \mathcal{A}_L$. Otherwise, without loss of generality, assume that L has a smaller wave number than L'. In other words, denote two wave numbers of L and L' as w and w', respectively, then $w < w'$. Since L is directly committed, L must be referenced by at least $f+1$ blocks in round $\langle w, 1\rangle$, with these blocks being denoted by $\mathcal{R}$. Besides, as each block in round $\langle w, 2\rangle$ references at least $2f+1$ blocks in round $\langle w, 1\rangle$, at least one of these blocks must be a member of $\mathcal{R}$. Therefore, each block in round $\langle w, 2\rangle$ must indirectly reference L. Similarly, each block in a round with the round number larger than $\langle w, 2\rangle$ must indirectly reference L. Thus, L' must reference L and $L \in \mathcal{A}_{L'}$, which concludes the proof. □

LEMMA 4. *In a wave of LightDAG2, all blocks in the third round will not reference contradictory blocks in the first round.*

Proof. According to Rule 2 of LightDAG2, all blocks delivered in the second round will not directly reference contradictory blocks in the first round. A round in the third round will directly reference blocks in the second round and indirectly reference blocks in the first round. Thus, all blocks in the third round will not reference contradictory blocks in the first round. □

LEMMA 5. *In LightDAG2, if two candidate blocks (C and C') are directly committed by two non-faulty replicas, respectively, then either $C \in \mathcal{U}_{C'}$ or $C' \in \mathcal{U}_C$.*

Proof. If C and C' have the same wave number w, either C or C' will be referenced by at least $2f+1$ blocks in the third round of wave w. Besides, C and C' are in the same slot. According to Lemma 4, C and C' must be identical, and we have $C \in \mathcal{U}_{C'}$ and $C' \in \mathcal{U}_C$. Otherwise, without loss of generality, assume that C has a smaller wave number than C'. In other words, denote two wave numbers of C and C' as w and w', respectively, then $w < w'$. Since C is directly committed, C must be referenced by at least $2f$+1 blocks in round $\langle w, 2\rangle$, which correspond to at least $2f$+1 slots in round $\langle w, 2\rangle$. Among these slots, at least f+1 slots belong to non-faulty replicas and denote these slots as $\mathcal{T}$. On the other hand, as each block in round $\langle w$+1$, 0\rangle$ references blocks of at least $2f$+1 slots in round $\langle w, 2\rangle$, at least one of these slots must be a member of $\mathcal{T}$. Therefore, each block in round $\langle w$+1$, 0\rangle$ will indirectly reference C and consistently determine C as the unique block in C's slot. Since $w' \geq w$+1, C' will also determine C as the unique block in C's slot, thus $C \in \mathcal{U}_{C'}$. □

LEMMA 7. *In LightDAG2, if a reproposed block with a Byzantine proof for p_b is delivered in a wave w, any CBC block of wave $v (v > w)$ that directly references a block created by p_b will not be delivered.*

Proof. We assume by contradiction that both the reproposed block B and the CBC block C are delivered. Since a CBC block is delivered only if $2f$+1 replicas vote for this block, at least one non-faulty replica votes for both B and C, which is denoted as p_v. We consider the following two cases:

- **Case 1:** If p_v votes for B first, it must receive a Byzantine proof for p_b. According to Rule 3, it will not vote for any CBC block that references a block created by p_b. Therefore, p_v will not vote for C, a contradiction.
- **Case 2:** If p_v votes for C first, according to Rule 3, it will not vote for any CBC block in a wave with a smaller wave number than v. Therefore, it will not vote for B, a contradiction.

□

LEMMA 8. *In LightDAG2, if a reproposed block with a Byzantine proof for p_b is delivered in a wave w, each replica will recognize p_b's Byzantine identity in or before the round $\langle w+1, 2\rangle$.*

Proof. Once a reproposed block is delivered in wave w, at least $f+1$ non-faulty replicas must have voted for it and recognized the Byzantine identity of p_b, which are denoted by a set $\mathcal{R}$. According to Rule 3, each replica in $\mathcal{R}$ has not created a CBC block in wave $w+1$ and will contain the Byzantine proof in its created CBC block in round $\langle w+1, 1\rangle$. Denote the CBC blocks created by $\mathcal{R}$ in round $\langle w+1, 1\rangle$ as a set $\mathcal{S}$. Since each block in round $\langle w+1, 2\rangle$ will reference $2f+1$ blocks in

round $\langle w+1, 1\rangle$, at least one block in $\mathcal{S}$ will be referenced, and p_b's Byzantine identity will be recognized. □

LEMMA 9. *In a wave w of LightDAG2, if no reproposed block is delivered, the probability that a candidate block gets directly committed is larger than $\frac{1}{3}$.*

Proof. When the leader slot is revealed in wave w, at least $2f+1$ blocks in round $\langle w, 1\rangle$ are delivered, each of which references at least $2f+1$ blocks in round $\langle w, 0\rangle$. Since $(2f+1)\cdot(2f+1) > f\cdot n+(f+1)\cdot(f+1)$, there are more than $f+1$ blocks in round $\langle w, 0\rangle$, denoted by a set $\mathcal{M}$, each of which is directly referenced by at least $f+1$ blocks in round $\langle w, 1\rangle$. As no reproposed block is delivered, there is only one block in each slot in round $\langle w, 1\rangle$. Therefore, each block in $\mathcal{M}$ will be referenced by every block in round $\langle w, 2\rangle$. If the leader slot matches a block in $\mathcal{M}$, this block can be committed directly. The probability that a leader slot matches a block in $\mathcal{M}$ is at least $\frac{f+2}{3f+1} > \frac{1}{3}$, and the probability that a block gets directly committed in this case is larger than $\frac{1}{3}$. □