Consensus Mechanism Design based on Structured Directed Acyclic Graphs

Jiahao He[*], Guangju Wang[*], Guangyuan Zhang[*], and Jiheng Zhang[†]

[*†]DMAC Lab
[*†]Department of Industrial Engineering and Decision Analytics
[†]Department of Mathematics
[*†]The Hong Kong University of Science and Technology

January 11, 2021

Abstract

Capacity limit is a bottleneck for broader applications of blockchain systems. Scaling up capacity while preserving security and decentralization are major challenges in blockchain infrastructure design. In this paper, we design a PoW-based mechanism by endowing directed acyclic graphs (DAG) with a novel structure so that peers can reach consensus at a large scale. At a high level, we break large blocks into smaller ones to improve utilization of broadcast network and embed a Nakamoto chain inside the DAG in a decent way to ensure security. We further exploit the DAG structure and design a mempool transaction assignment method. The method reduces the probability that a transaction is processed by multiple miners and hence improves processing efficiency. Without sacrificing security and decentralization, our design significant scales up capacity and also addresses important issues such as high latency and mining power concentration in existing blockchain systems.

Key words: consensus; directed acyclic graph; proof of work; transaction assignment.

1 Introduction

Blockchain technology has achieved great success as exemplified by Bitcoin [9] and Ethereum [13]. They have proved that a decentralized and secure public ledger system is not only possible but also has a great impact on financial systems and gives rise to a whole new ecosystem with extendable services and applications. The foundation of Blockchain technology is a consensus mechanism built upon a chain structure where peers over the network can store information in a decentralized manner. Taking Bitcoin [9] for example, blocks containing transactions are connected in a chain with additional cryptographic requirements and Nakamoto consensus [9] based on the chain is able to achieve both security and decentralization.

With its rapid adoption over the last decade, blockchain technology has now come face to face with a serious bottleneck, the extremely limited capacity, i.e., a small number of transactions per second (TPS). Removing this bottleneck will be a significant breakthrough in advancing blockchain technology, and open the possibility for a wide range of applications. However, scaling up the capacity should not be at any compromise of security or decentralization. Along with the capacity limit, the following issues are also of major concern in current blockchain systems.

1. **Latency.** It usually takes a long time for a transaction to be securely recorded on a Blockchain.

2. **Transaction Fee.** Since miners are self-interested, they would try to pack transactions with high fees into their blocks to maximize their revenue. Thus, transactions with little to no transaction fees hardly have a chance to be processed.

3. **Concentration of mining power within mining pools.** The huge fluctuations of the actual reward around the expectation force most miners to join mining pools in order to smooth their income. Consequently, hashing power are concentrated within a few large mining pools.

Confined in a chain structure, we can try to increase the block size, to decrease the time interval between blocks, or to use a combination thereof to increase the TPS. However, since larger blocks require more time to propagate over a peer-to-peer network, such an attempt would increase the occurrence of forked blocks and hence can only expand capacity to a limited extent, even in the absence of malicious miners. Thus, to increase the capacity by an order of magnitude, we must explore beyond the chain structure. This naturally leads to the idea of expanding the chain of blocks to a directed acyclic graph (DAG) of blocks to allow *parallelism*. The idea has been implemented by IOTA [11], Conflux [8], CoDAG [3], BlockDAG [4], and BlocHIE [6]. However, in IOTA, the block is not fully and ordered and the security is not guaranteed. Although researchers develop some other DAG-based protocol, such as Conflux [8], their designs are complicated and need to calculate the "weight" for each block to reach consensus. The weight update approaches such as Ghost [12] suffer from what is known as the "balancing attack" [1]. In this paper, we design a DAG-based structure that can incorporate the well-known Nakamoto chain to ensure consensus as well as a simple miner workflow without calculating the "weight".

To scale up capacity, we break a large block into multiple smaller ones with smaller sizes, fewer reward, and much easier cryptographic puzzles. To ensure consensus, our key idea is to embed a Nakamoto chain in a DAG by designing a structure that is strongly connected and incorporates miner information. Blocks on the embedded Nakamoto chain are *milestones* and are harder to mine than *regular blocks*. However, the mining workflow for both types of blocks are the same and before PoW is completed, the creator of a block does not know in advance whether the block will become a milestone or not. More specifically, before PoW starts, a miner specifies three pointers as follows. The first pointer points to the miner's previous block.

This pointer links all blocks mined by a miner into a *peer chain*[1] which encodes the miner's information such as his hashing power. The second pointer points to the latest milestone on the longest milestone chain. It ensures that, if the block does turn out to be a milestone *after* PoW, all milestones will remain connected through their second pointers. The third pointer points to a block mined by some other miner to enhance connectivity among peer chains. Figure 1 illustrates the DAG structure induced from this connecting rule. Having specified the pointers, the miner simply tunes nonce until the hash of the block exhibits a required pattern, e.g., the leading 10 bits are 0, in which case a regular block is produced. If by luck, the hash exhibits a more strict pattern, e.g., the leading 15 bits are all 0, then the block is qualified as a milestone. Note that the time to provide a valid PoW (either for a regular block or a milestone) can be approximately described by a memoryless exponential random variable, i.e., once a PoW for a regular block is found, the expected extra time to find a PoW for the current block to be a milestone is the same as the expected total time to find a PoW for a new block to be a milestone. Hence, miners will move on to create a new block once a PoW for a regular block is found and will not devote to producing milestones.

Our design successfully addresses the capacity issue as well as the three additional issues listed above without sacrificing security or decentralization. With blocks of small size, miners can broadcast transactions (stored in small blocks) continuously over time, instead of accumulating large transaction batches and broadcast every once in a while. Smaller block with easier hashing requirement also greatly reduces the variability in mining rewards, thus eliminating miners' need to rely on mining pools to smooth their income and discouraging the need for large mining pool. To see our design is secure, note that the growth of the milestone chain is identical to that of a Nakamoto chain. Hence, the proven security of Nakamoto chain immediately extends to the that of all blocks pointed (directly or recursively) by the milestone chain, which is at a large scale due to the strong connectivity provided by the third pointer. Moreover, the DAG also provides an ordering of transactions so that all honest peers (miners) will be able to build the same public ledger once they have reached consensus on the DAG. Furthermore, the miner information encoded in peer chains enables us to design a transaction assignment method. We probabilistically assign transactions to miners based on their peer chain states to reduce the probability that a transaction is processed by multiple miners. This method further improves processing efficiency and reduces latency, especially for those transactions with little transaction fees.

Our work can be viewed as an extension of blockchain technology, such as Bitcoin [9] and Ethereum [13], and consensus mechanism on DAGs, such as IOTA [11]. The rest of the paper is organized as follows. We describe in detail the proposed DAG structure and a protocol for decentralized maintenance of the structure in Section 3 and Section 4, respectively. In Section 5, we present a method to build a public ledger from the DAG and a reward scheme that incentivizes honest miners work as expected. We provide theoretical analysis as well as

[1]We use the terms"miner" and "peer" almost interchangeably. The only subtle difference is that a miner must solve cryptographic puzzles to create blocks, while a peer does not. Like a miner, a peer may propagate blocks to a peer-to-peer network.

numerical simulation to evaluate the performance of our design in Section 6.1.

2 Literature Review

IOTA [11] is the first one that implements the idea of DAG and achieves a good throughput performance. However, the security of IOTA is not guaranteed and the transactions in IOTA are not fully ordered. Recently, some DAG-based systems try to solve these issues in IOTA. BlockDAG [4] provides an algorithm to determine the order of a DAG. Conflux [8] solves the waste of discarding the fork blocks by selecting the "pivot chain" with the highest "weight" whose consensus is guaranteed by Ghost [12]. And the full order of transaction in Conflux is kept by maintaining the "pivot chain". CoDAG [3] splits the graph into different levels, and selects blocks in each level by computing their "connectivity". Comparing with their designs, we eliminate the need to compute "weights" or "connectivity" which may be time-consuming and our security is guaranteed by Nakamoto consensus. Some other works design DAG-based blockchain systems for some specific applications. For instances, LDV [14] designs a lightweight blockchain system for Vehicular Social Network based on DAG structure; BlocHIE [6] uses two loosely coupled blockchain to store and share different kinds of medical data. There are other ways to reach consensus without using DAG structure and PoW. For instance, Algorand [2] provides a new scaling consensus based on Byzantine Agreement, and Thunderella [10] offers a new consensus mechanism that is robust in the worst case and enables fast transaction confirmation in the optimistic case. These topics are beyond the scope of this paper.

3 The Proposed Structure in a DAG

In this section, we describe in detail our proposed structure in a DAG. The objective of this structure is to enable miners across the network to reach a consensus on a set of data, so that they can build and maintain a public ledger in a distributed fashion that all honest peers agrees on. We present the DAG structure in a general setting without dependency on public ledgers, since it can potentially have other applications. We first define the basic elements, namely blocks, then we introduce a few concepts to describe the structure of the DAG formed by blocks.

3.1 Blocks

Blocks are basic elements in the DAG. As a basic unit of information, a block `B` contains a `message`, which is the core data needed for particular applications, e.g., transactions for cryptocurrency applications. It also contains additional information, namely the *block header*, for integrity checking and positioning in the DAG. Assume there is a random oracle H which maps a string of arbitrary length to a unique identity. In actual implementation, we would use a reasonably good cryptographic hash function, e.g. `SHA-256`, as the random oracle. We apply the random oracle to the concatenation of all parts of a block, symbolically denoted by $H(\texttt{B})$,

to obtain the block's *identity*. By a block's *integrity*, we mean that the block and its identity match under the random oracle. Formally, a block

$$\mathtt{B} = (\mathtt{id}_{\mathtt{prev}}, \mathtt{id}_{\mathtt{ms}}, \mathtt{id}_{\mathtt{tip}}, \mathtt{peer}, \mathtt{nonce}, \mathtt{message}), \tag{1}$$

where ($\mathtt{id}_{\mathtt{prev}}, \mathtt{id}_{\mathtt{ms}}$, $\mathtt{id}_{\mathtt{tip}}$) are the unique identities of some other blocks in the DAG, `peer` is the miner who creates this block, and `nonce` is the solution to the cryptographic puzzle. In the language of graph theory, blocks form the set of vertices and each pointer corresponds to a directed edge. Among all blocks, there is a special block, namely *the genesis* `0`, which contains certain trusted setup information and is the unique root of the whole DAG.

3.1.1 Proof of Work and Block Types

We require a proof of work (PoW) for each block to decide if a miner has the right to form the block and to determine the type of a block. Suppose the hash result $H(\mathtt{B})$ is a string of zero-one bits and denote by $.H(\mathtt{B})$ the number in $[0, 1]$ to which this string of bits converts following the convention in [2]. Since H is a random oracle, $.H(\mathtt{B})$ can be modeled as a random variable uniformly distributed on the interval [0,1] for any block *a priori*. In order to "prove work", a miner needs to vary the `nonce` until $.H(\mathtt{B})$ exhibits a required pattern. For the ease of presentation, we require that $.H(\mathtt{B}) < d$, where d reflects the *difficulty* of finding such a nonce. Among all the blocks, a portion of them to are *milestones*. A block is a milestone if $.H(\mathtt{B}) < pd$, where p indicates the probability that a block is a milestone. Note that a miner has to specify all parts of a block including the three pointers, the main message and the nonce before computing the hash. The type of block is revealed only after the peer has worked on it. In other words, a peer does not know whether he is working toward a milestone or a regular block. A miner may continue working on the block after obtaining a hash $.H(\mathtt{B}) \in [pd, d)$ with the intention of making it a milestone. However, devoting a miner's hashing power to the creation of milestone blocks will not change the expected number of milestone blocks he is able to create. Since a regular block also yields a mining reward as we specified in Section 5, albeit a smaller reward than that offered by a milestone, and involves a transaction fee, exclusively mining milestone blocks is not economically beneficial for peers.

3.1.2 The Pointers

The pointers specify the relation among blocks. We now state requirements for the three pointers ($\mathtt{id}_{\mathtt{prev}}, \mathtt{id}_{\mathtt{ms}}, \mathtt{id}_{\mathtt{tip}}$) and discuss the induced structure under these requirements. Since all precedents of a particular block can be found recursively through the pointers, consensus on the block can be extended to all its precedents. Thus, we will say that a block `B` *confirms* another $\mathtt{B}'$ if $\mathtt{B}'$ is a precedent of `B`, i.e., there is directed path from `B` to $\mathtt{B}'$. See (10) for a mathematical definition of confirming previous blocks.

$\mathtt{id}_{\mathtt{prev}}$ and Peer Chains. The first pointer $\mathtt{id}_{\mathtt{prev}}$ points to the most recent block created by the same miner or the genesis if the miner has not mined any blocks before. Under this

requirement, blocks mined by the same miner are organized into a chain, namely the *peer chain*. The *head* of the chain, defined to be the most recent block, can also be interpreted as the state of the miner. A peer chain is designed to provide not only a clear structure in the DAG, but also valuable information including the miner's mining history, from which we can estimate the miner's hashing power. In Section 5, we will introduce a transaction scheduling scheme that utilizes miners' states and estimated hashing power to reduce the probability that a transaction is processed by multiple miners. The peer chain could also potentially generalize our design to credit-based applications. Formally, we require

$$\mathtt{B}.\mathtt{id}_{\mathtt{prev}} = H(\mathtt{B}') \implies \{\mathtt{B}'.\mathtt{peer} = \mathtt{B}.\mathtt{peer}\} \text{ or } \{\mathtt{B}' = \mathtt{0}\}. \tag{2}$$

Note that the mathematical requirement cannot prevent a miner from forking his own peer chain (e.g., not appending to the miner's most recent block). However, such behavior cannot affect the consensus. Moreover, under the reward scheme introduced in Section 5, a miner will waste his hashing power and gain less mining reward if he forks his own peer chain. In fact, incentivizing a miner to append a newly mined block to his most recent block lessens the effect of "lazy connecting", i.e. not appending to recent blocks, an issue raised in the implementation of IOTA [11].

$\mathtt{id}_{\mathtt{tip}}$ and Connectivity. To create connectivity among different peer chains, we use $\mathtt{id}_{\mathtt{tip}}$ to point to a regular block of another miner. The genesis $\mathtt{0}$ is set as the default when no such block exists. Formally,

$$\mathtt{B}.\mathtt{id}_{\mathtt{tip}} = H(\mathtt{B}_t) \implies \{\mathtt{B}_t.\mathtt{peer} \neq \mathtt{B}.\mathtt{peer},\ .H(\mathtt{B}_t) \in [pd, d)\} \text{ or } \{\mathtt{B}_t = \mathtt{0}\}. \tag{3}$$

Again, the directed edge from block $\mathtt{B}$ to $\mathtt{B}_t$ plays a role of confirming $\mathtt{B}_t$ and all of the blocks that $\mathtt{B}_t$ confirms. Intuitively, a stronger connection leads to a faster confirmation, thus we could have multiple such pointers to further enhance the connectivity. However, these pointers have to be synchronized, verified and stored by all peers and therefore having more pointers will incur a higher overhead cost. Our analysis in Section 6.1.3 shows that having one such pointer is enough to ensure reasonable confirmation latency.

$\mathtt{id}_{\mathtt{ms}}$ and an Embedded Milestone Chain. The pointer $\mathtt{id}_{\mathtt{ms}}$ points to a milestone block $\mathtt{B}_m$ or the genesis $\mathtt{0}$, i.e.

$$\mathtt{B}.\mathtt{id}_{\mathtt{ms}} = H(\mathtt{B}_m) \implies \{.H(\mathtt{B}_m) < pd\} \text{ or } \{\mathtt{B}_m = \mathtt{0}\}. \tag{4}$$

Our milestone chain works in the same way as the Nakamoto chain, the backbone of Bitcoin blockchain. The difference is that each block on our milestone chain is smaller in size and confirms some other regular blocks in a structured way. A major barrier to scaling up the capacity of a blockchain system is the slow synchronization of large blocks in a peer-to-peer network. Our milestone chain, consists of much smaller blocks, is designed as a bridge between high throughput and stable synchronization. Each milestone, despite being small, confirms a relatively large number of other blocks, as formally defined in (10) below. In other words, each

milestone confirms a part of history. Therefore, as long as the peers reach a consensus on all milestones, they reach a consensus on the entire history. To achieve the objective, we have to put all milestones in a chain structure by requiring all blocks to have a pointer pointing to a previous milestone. This is because when preparing a block, in particular when setting the pointer $\mathtt{id}_{\mathtt{ms}}$, there is no way of knowing whether or not that block will be a milestone until cryptographic puzzle is solved. If the block turns out to be a milestone, we want to make sure that it connects to a previous milestone via the pointer $\mathtt{id}_{\mathtt{ms}}$.

3.2 A Structured DAG

Now we introduce a few concepts and notations that help us to describe the structure of the DAG and the relation among blocks resulting from the PoW requirements and the pointer requirements above. Let $\mathcal{G}$ be a collection of blocks including the genesis $\mathtt{0}$, such that each non-genesis block satisfies (2)–(4), and

$$\forall\ \mathtt{B} \in \mathcal{G}, H(\mathtt{B}') = \mathtt{B}.\mathtt{id}_{\mathtt{key}},\ \mathtt{key} \in \{\mathtt{prev}, \mathtt{ms}, \mathtt{tip}\} \implies \mathtt{B}' \in \mathcal{G}. \tag{5}$$

In other words, all blocks to which $\mathtt{B} \in \mathcal{G}$ points are also in $\mathcal{G}$. $\mathcal{G}$ is indeed a directed graph with blocks as vertices and pointers as directed edges. A *path* from $\mathtt{B}$ to $\mathtt{B}'$ is a consistently-directed sequence of edges from $\mathtt{B}$ to $\mathtt{B}'$. By construction, $\mathcal{G}$ is also *acyclic*, i.e., there is no directed path from $\mathtt{B}$ to itself for any $\mathtt{B}$ in $\mathcal{G}$.

We say that a *block* $\mathtt{B}$ is *syntactically valid* if its format satisfies (1) and $.H(\mathtt{B}) < d$. A *DAG* $\mathcal{G}$ is *syntactically valid* if all of its blocks are syntactically valid, and it is acyclic and satisfies (5). A block $\mathtt{B}$ is *syntactically valid for* $\mathcal{G}$ if $\mathcal{G} \cup \{\mathtt{B}\}$ is syntactically valid. In the decentralized maintenance of the DAG, one of a peer's basic task is to ensure his local DAG is syntactically valid. Such a check protects the system from being flooded with invalid blocks.

It will become self-evident that under our protocol, the milestone tree grows in the same way as the tree occurring in a blockchain with forks. The milestone tree plays an essential role in connecting all of the peer chains, while the pointers $\mathtt{id}_{\mathtt{tip}}$ further enhance the connectivity of the DAG Figure 1 provides an illustration of such a structure.

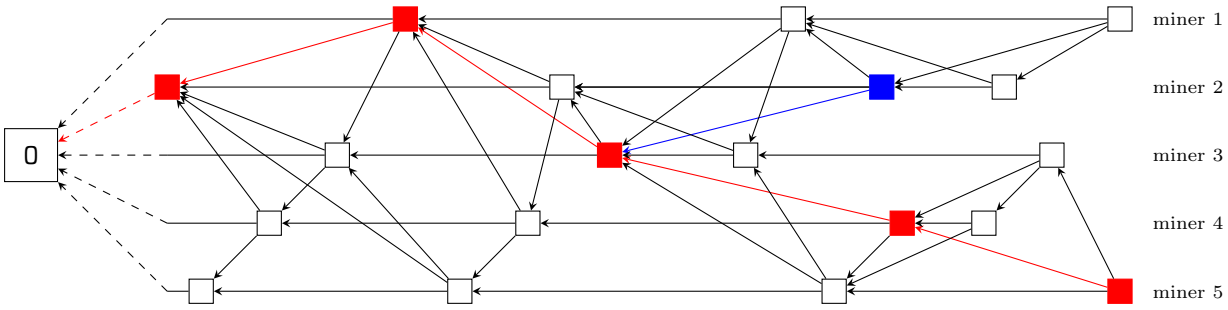

Figure 1: DAG structure for 5 miners with milestone forks.

We now define the *height of milestone blocks* in a structured DAG. First, the height of the genesis $\eta(\mathtt{0}) = 0$, and inductively, for a milestone block $\mathtt{B}$, i.e., $H(\mathtt{B}) < pd$, the height is defined

as

$$\eta(\mathtt{B}) = \eta(\mathtt{B}_m) + 1, \quad \text{where } H(\mathtt{B}) < pd \text{ and } \mathtt{B}.\mathtt{id}_{\mathtt{ms}} = H(\mathtt{B}_m). \tag{6}$$

Thus, the height of a milestone block is imply its depth in the tree consisting of the genesis and all milestone blocks in $\mathcal{G}$. Formally, the *leaf set* of all milestone blocks is

$$\begin{aligned}\mathcal{T}_m(\mathcal{G}) = \{\mathtt{B}_m \in \mathcal{G} : .H(\mathtt{B}_m) < pd \text{ and} \\ \nexists\, \mathtt{B}'_m \in \mathcal{G} \text{ s.t. } \mathtt{B}'_m.\mathtt{id}_{\mathtt{ms}} = H(\mathtt{B}_m) \text{ and } .H(\mathtt{B}'_m) < pd\}.\end{aligned} \tag{7}$$

For a milestone block $\mathtt{B}_m$ in the leaf set with $\eta(\mathtt{B}_m) = n$, following the pointers $\mathtt{id}_{\mathtt{ms}}$ we can find a unique sequence of blocks

$$\mathtt{B}_{m,n} = \mathtt{B}_m, \mathtt{B}_{m,n-1}, \mathtt{B}_{m,n-2}, ..., \mathtt{B}_{m,1}, \mathtt{B}_{m,0} = \mathtt{0}, \tag{8}$$

such that $.H(\mathtt{B}_{m,k}) < pd$ and $\mathtt{B}_{m,k}.\mathtt{id}_{\mathtt{ms}} = H(\mathtt{B}_{m,k-1})$ for all $k = 1, 2, \ldots, n$. We call this sequence the *milestone chain for block* $\mathtt{B}_m$. So each block in the leaf set of all milestone blocks represents a milestone chain with length equal to the height of that block. We define the height of the DAG to be the length of the longest chain(s),

$$\eta(\mathcal{G}) = \max\{\eta(\mathtt{B}) : \mathtt{B} \in \mathcal{T}_m(\mathcal{G})\}. \tag{9}$$

Note that multiple longest chains might exist. To choose a longest chain, we just need to choose a block with the largest height in the leaf set. The nth milestone is defined to be the block on the longest milestone chain with height n.

For any milestone block $\mathtt{B}_m \in \mathcal{G}$, we define the *DAG confirmed by the milestone* $\mathtt{B}_m$ to be

$$\mathcal{C}(\mathtt{B}_m) = \{\mathtt{B} \in \mathcal{G} : \text{ there exits a path from } \mathtt{B} \text{ to } \mathtt{B}_m\} \cup \{\mathtt{B}_m\}. \tag{10}$$

If $\mathtt{B}'_m \in \mathcal{G}$ is the milestone or genesis that immediately preceding $\mathtt{B}_m$, i.e. $\mathtt{B}_m.\mathtt{id}_{\mathtt{ms}} = H(\mathtt{B}'_m)$, then we define the *level set* to be

$$\mathcal{S}(\mathtt{B}_m, \mathtt{B}'_m) = \mathcal{C}(\mathtt{B}_m) \setminus \mathcal{C}(\mathtt{B}'_m). \tag{11}$$

With slight extension of notation, we define $\mathcal{C}(k) = \mathcal{C}(\mathtt{B}_{m,k})$ to be the DAG confirmed by the kth milestone and $\mathcal{S}(k) = \mathcal{C}(k) \setminus \mathcal{C}(k-1)$ to be the kth level set, with $\mathcal{C}(0) = \{\mathtt{0}\}$. In Figure 2, level sets $\mathcal{S}(k)$, $\mathcal{S}(k+1)$, $\mathcal{S}(k+2)$ and $\mathcal{S}(k+3)$ are shaded in gray. The *pending set* is defined to be the set of all blocks that have not yet been confirmed by any milestone on the longest milestone chain. It is worth pointing out that the blue block is also a milestone block. This milestone forked from the $(k+1)$th milestone due to either network synchronization delay or attack. The blue and red blocks in Figure 2 form the milestone tree above defined. The blue block and the current $(k+2)$th milestone will compete for confirmations by future milestones based on Nakamoto consensus. If the blue blocks fails, i.e. all honest peers thinks it is a fork, it will be treated as a regular block in Section 5 when we build the ledger.

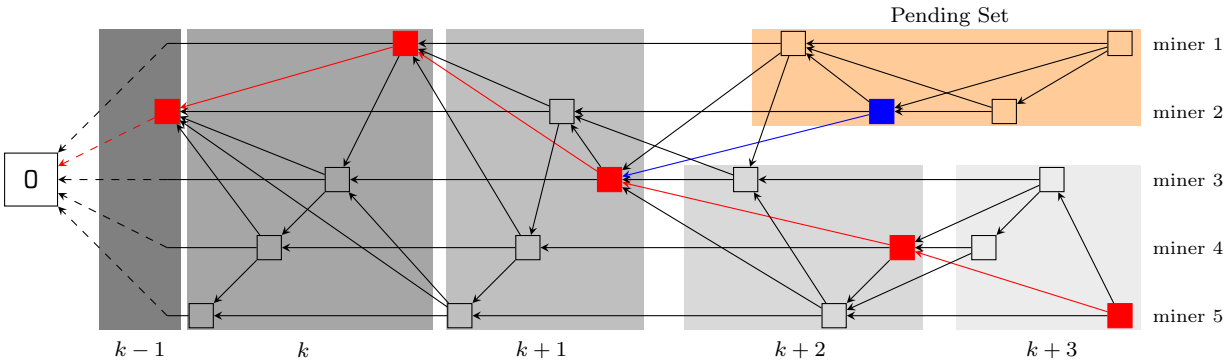

Figure 2: An Illustration of Level Sets and the Pending Set

4 Protocol for Maintaining a Local DAG

The structured DAG is collectively created and maintained in a distributed fashion by all peers over a peer-to-peer network following a protocol. Peers obeying this protocol are said to be *honest* and peers not following this protocol are said to be *malicious*. In a decentralized system, each peer will have his own local DAG, which can be different from the local DAG of another peer due to issues such as network synchronization delay and malicious attacks. This can be illustrated by comparing Figure 2 and Figure 3. Suppose that due to network synchronization, an honest peer has not received the latest milestone created by miner 5. He may then regard the blue milestone as the $(k+2)$th on the longest chain. In this case, his local view is depicted in Figure 3.

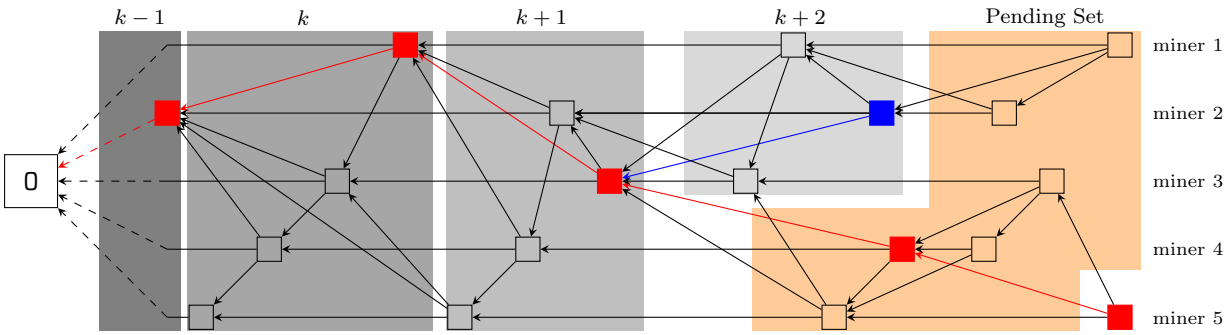

Figure 3: Level Sets from a Different Perspective

For a peer a, denote his local DAG by $\mathcal{G}_a$, which is evolving as he creates and receives blocks. All miners start their peer chains from the genesis block 0. We now describe the protocol to be followed by each peer/miner. The objective is that all honest peers agree upon the same DAG (see Section 6.1.1), and consequently to use the same set of data to construct the public ledger. We will present our protocol in the setting of a cryptocurrency application for concreteness and clarity, although the protocol can potentially be generalized to other applications. To this end, we first discuss the mempool, which is the buffer holding all pending transactions to be processed.

4.1 Mempool Transaction Assignment

When someone initiates a transaction, he will broadcast it to a few connected peers. This is implemented via an inventory message and getdata approach in Bitcoin for example. If a peer considers the transaction valid after receiving it, he will also broadcast the transaction to all of his peers. A peer also need to remove a transaction from his local mempool once he find out that transaction has already been put in a block, created either by himself or other peers. In such a way, all peers will locally have a buffer of all outstanding transactions waiting to be processed.

The purpose of expanding the chain structure to a DAG structure is to allow parallelism for scaling up the capacity. However, with the current mempool design, it is highly possible that the same transaction, especially when it is associated with a high transaction fee, will be processed by multiple miners due to network broadcast delay. Consequently, the same transaction may end up in multiple blocks and much capacity would go to waste. If someone with malicious intent wants to ruin the network by wasting a large portion of the capacity, he could simply broadcast transactions with attractive fees in rapid succession. Such an attack would be much less costly than owning and using a certain percentage of the hashing power, and can effectively reduce the capacity.

Therefore, we need to design a mechanism to effectively reduce the chance of *collision*, i.e., one transaction being processed by multiple miners. The basic idea is to dynamically limit the number of transactions a miner can process at any time by using the hash result $.H(H(\mathtt{B}_i), \mathtt{Tx})$ as a distance between transaction $\mathtt{Tx}$ and the miner's state $\mathtt{B}_i$, i.e. the most recent block on miner i's peer chain. The peer chain can also give a clear count of the proportion of blocks created by this miner in any level set, and thus it provides a reference of the miner's hashing power. Suppose miner i possesses $q_i \in [0, 1]$ proportion of the total hashing power. We say that transaction $\mathtt{Tx}$ is *workable* for miner i if and only if the distance

$$.H(H(\mathtt{B}_i), \mathtt{Tx}) \leq cq_i, \tag{12}$$

where c is hyper-parameter to be specified based on experiment or careful analysis as illustrated in Section 6.1. This implies that a transaction is workable for miner i with probability $\min(cq_i, 1)$ at any time. Note that disguising as multiple peers by creating multiple peer chains cannot increase a miner's workable transactions, as the miner has to split his hashing power over the peer chains so that each peer chain will have a smaller number of workable transactions. When no transaction is workable for a miner, we allow the miner to produce an empty block that contains no transaction, which refreshes his own state and helps harden the PoW level on the whole DAG.

We now provide some simple calculation to show how this method is able to reduce collision. First, miner i can surely process all transactions if c is chosen such that $cq_i \geq 1$. So we just

limit our choice of c to the range $(0, 1/\max_i q_i)$. When n is large, we have

$$\mathbb{P}(\texttt{Tx} \text{ is not workable for any miner}) = \prod_{i=1}^{n}(1 - cq_i) \leq \left(\frac{\sum_{i=1}^{n}(1 - cq_i)}{n}\right)^n \approx e^{-c},$$

$$\mathbb{P}(\texttt{Tx} \text{ is workable for exactly one miner}) = \sum_{i=1}^{n} cq_i \prod_{j\neq i}(1 - cq_j) \approx e^{-c}\sum_{i=1}^{n}\frac{cq_i}{1 - cq_i} \approx ce^{-c},$$

$$\mathbb{P}(\texttt{Tx} \text{ is workable by more than one miners}) \approx 1 - e^{-c} - ce^{-c}.$$

Thus, a small c helps to reduce collision. For example, $c = 0.1$ results in a collision probability less than 0.5%. It may be of concern that with a small c, a transaction cannot be processed by any miner. However, note that a miner's state changes every time he creates a new block and hence, the set of workable transactions for him also changes dynamically. Therefore, a transaction will become workable for some miner after a few rounds of state change. To find an appropriate c to strike a balance between the wasted capacity due to collision and the extra waiting time for a transaction to become workable, we perform a detailed analysis via a modeling approach in Sections 6.1.2 and 6.1.3.

4.2 Receiving a Block

Suppose that before adding a new block, peer a locally stores $(\mathcal{G}_a, \texttt{B}_m)$ where $\mathcal{G}_a$ is a syntactically valid DAG and $\texttt{B}_m \in \mathcal{G}$ is one of the highest milestone block. A peer may be new to the network, i.e., $(\mathcal{G}_a, \texttt{B}_m) = (\{\texttt{0}\}, \texttt{0})$, or has been offline for a while. In this case, the peer should first learn from his connected peers the height of the longest milestone chain. If the height of his local DAG is much smaller, he should start downloading the missing level sets. Once a peer's local height is close to the heights of his connected peers, e.g., within a threshold specified in our code implementation, the peer can start to receive broadcasted newly mined blocks. Suppose that peer a receives a block $\texttt{B}$ through broadcast. It is possible that he does not have some blocks to which $\texttt{B}$ either directly or indirectly points due to synchronization delay. He should start a process to *solidify* the block $\texttt{B}$ by asking his peers for the missing blocks from its peers. Once he has obtained all the missing blocks, peer a needs perform topological sorting (e.g. Section 2.2.3 in [7]) of these blocks, so that blocks can be added sequentially to the peer's local DAG.

We now describe how to add a block $\texttt{B}$, when the blocks to which it points are already in the DAG $\mathcal{G}_a$. First, check if block $\texttt{B}$ is syntactically valid for $\mathcal{G}_a$. If not, discard it immediately; otherwise update as

$$(\mathcal{G}_a, \texttt{B}_m) \leftarrow \begin{cases} (\mathcal{G}_a \cup \{\texttt{B}\}, \texttt{B}), & \text{if } .H(\texttt{B}) < pd \text{ and } \eta(\texttt{B}) > \eta(\mathcal{G}_a), \\ (\mathcal{G}_a \cup \{\texttt{B}\}, \texttt{B}_m), & \text{otherwise,} \end{cases}$$

and relay the block $\texttt{B}$ to connected peers. If block $\texttt{B}$ contains a transaction that is in the peer's mempool, the peer removes the transaction from his mempool. We do not require the peer to perform a major update even if $\texttt{B}$ is a higher milestone than $\texttt{B}_m$ and is on a milestone chain different from that of $\texttt{B}_m$. As long as miners keep track of the highest milestone and points to it with $\texttt{id}_{\texttt{ms}}$ when creating new a block, they can reach consensus on the longest chain and the

system is secure. Thus, we simply keep all blocks on the forked milestone chain, allow them to be pointed by later blocks with whichever appropriate pointers, and treat them as regular blocks when we translate the DAG into a public ledger. Note that we do not require the peer to verify the transaction in block $\mathtt{B}$ at this stage. Taking Bitcoin for example, the verification requires verifying signatures and checking that all inputs of the transaction correspond to some unspent outputs in the public ledger according to the current local DAG, and ensuring no double spending occurs among all transactions. We postpone such verification to the moment when we convert the local DAG to a ledger, primarily because it takes time, in particular the secure latency (Section 6.1.3), to reach consensus on the DAG.

4.3 Creating a block

Suppose that before creating a new block, miner a locally stores $(\mathcal{G}_a, \mathtt{B}_m, \mathtt{B}_a)$ where $\mathcal{G}_a$ is a syntactically valid DAG, $\mathtt{B}_m$ is one of the highest milestone block in $\mathcal{G}_a$, and $\mathtt{B}_a$ is the genesis or the most recent block on a's peer chain. If the miner wants to create a block $\mathtt{B}$, and have it accepted by all other peers as part of their local DAGs, the actions to be taken are as follows.

1. **Evaluate** `message`**.** The miner first searches for a transaction from his mempool that is
   (a) workable with respect to $\mathtt{B}_a$ as required in by (12),
   (b) compatible with the miner's ledger, i.e. the input(s) for this transaction should be in the unspent outputs of the ledger constructed based on the DAG $\mathcal{C}(\mathtt{B}_m)$.

   If such a transaction exists, set `message` to be the transaction; otherwise, set `message` to be empty.

2. **Set Pointers.**
   (a) Set $\mathtt{B}.\mathtt{id}_{\mathtt{ms}} = H(\mathtt{B}_m)$.
   Note that $\mathtt{B}_m$ is the highest milestone in the local DAG $\mathcal{G}_a$. However, due to network synchronization, $\mathtt{B}_m$ is may not necessarily be the highest milestone block in reality.
   (b) Set $\mathtt{B}.\mathtt{id}_{\mathtt{prev}} = H(\mathtt{B}_a)$.
   Note that block $\mathtt{B}_a$ and $\mathtt{B}$ being consecutive blocks created by the same miner is a stronger requirement than (2). See further discussion in the next section.
   (c) Define the *tip set* to be the set of regular blocks to which none of the blocks in $\mathcal{G}_a$ points and are not created by the miner a. Randomly pick a block $\mathtt{B}_t$ from the tip set and set $\mathtt{B}.\mathtt{id}_{\mathtt{tip}} = H(\mathtt{B}_t)$. If the tip set is $\{\mathtt{0}\}$ or empty, then set $\mathtt{B}.\mathtt{id}_{\mathtt{tip}} = H(\mathtt{0})$. This means that the miner needs to find a regular block $\mathtt{B}_t$ created by some other miner.

3. **Proof of Work.** Repeatedly change `nonce` and apply H to this block until $.H(B) < d$.

4. **Broadcast.** Finally, the miner needs to broadcast $\mathtt{B}$ through a peer-to-peer network. This certainly takes time, as we will discuss in the subsection on the broadcast delay assumption in Section 6.1.

5 Building a Public Ledger

We present this section in the context of the unspent transaction output (UTXO) model of Bitcoin for concreteness. The idea of constructing a public ledger from the DAG structure can be extended to other models. In the UTXO model, a *transaction* essentially specifies previous transaction outputs as new transaction inputs and allocates all input values to new outputs. Obviously, signatures are required in order to use an output as a transaction input. Readers can refer to Bitcoin documents for details. A *ledger* is an ordered list of transactions, from which we can construct the set of UTXOs. We say that a transaction is *valid* if it passes the signature validation and its inputs are indeed from the UTXOs according to all preceding transactions. A ledger is valid if all of its transactions are valid.

Since we do not impose validity checks when receiving transactions packed in blocks, there is no guarantee that all of the transactions in a local DAG are valid. For example, it is likely two or more double-spending transactions coexist in a local DAG. The level sets in our DAG along the longest milestone chain provide the natural ordering of level sets from low to high. Within each level set, we utilize a depth-first search algorithm to order all its transactions. In this way, as long as peers agree on the same DAG, they agree on the same way to order the same set of transactions. We then specify an algorithm to chose a subset of this transaction following the same ordering to form the public ledger.

Recall the definition of syntactical validity of our DAG, which every honest peer shall check when updating it. Although it specifies a structure in the DAG, it does not enforce hard constraints on the following requirements:

- Each miner should keep the blocks created by himself in a chain structure without any forks.
- A new block shall point to the most recent milestone being created.
- A new block shall point to a recent regular block on another peer chain when created.

In fact, we could not enforce hard constraints in most cases since there is synchronization takes time. For example, ideally, we would like to force each miner to connect to a block on another peer chain in the pending set. But as illustrated in Figures 2 and 3, there may be no consensus on the pending set due to temporary broadcast delay. Also, forbidding forking of a peer chain add too many complicated rules while performing syntactical validity check and is not easy to reach consensus when there is a forking attack. On the other side, not following the above three principles decreases the performance of our system in terms of latency and wasted capacity. For example, connecting to a too old regular block on another peer chain contributes very little to the connectivity, thus slow down the confirmation by a milestone block. Also, not connecting to the latest milestone creates the same issue of forking as in Bitcoin. Though eventually honest peer will reach consensus on the longest milestone chain, forking is not desirable as it is a waste of mined milestone blocks and cause delay in secure confirmation.

Reward Scheme. We design the following reward scheme to incentivize miners to follow the aforementioned principles. A miner should be rewarded by a fixed fee (even if the block does not contain a transaction) plus the transaction fee for creating a block. Exactly how much a miner can get out of a block depends on three factors: the type of the block, the ultimate status of this block, and the validity of the transaction in this block. The first two factors depends on consensus dictated by the longest milestone chain, and the last one depends on the ordering of transactions as we will explain later.

We have been using milestone and regular as block types so far. As discussed in Section 3, all the milestone blocks form a tree. When computing the reward, we would like to treat milestone block not on the longest chain as regular blocks. For ease of reference, let's call regular and forked milestone blocks *regular+*. Let r_m and r_n denote the fixed reward for milestones on the longest milestone chain and regular+ blocks, respectively. It is natural to set $r_m > r_n$ since milestone blocks are more difficult to create. Let $\mathtt{B}_m$ and $\mathtt{B}'_m$ be two consecutive milestones on the longest milestone chain with $\mathtt{B}_m.\mathtt{id}_{\mathtt{ms}} = H(\mathtt{B}'_m)$. Denote by

$$R(\mathtt{B}_m) = r_n(|\mathcal{S}(\mathtt{B}_m, \mathtt{B}'_m)| - 1)$$

the total amount of regular+ block rewards in the level set $\mathcal{S}(\mathtt{B}_m, \mathtt{B}'_m)$. We summarize our reward scheme in the following table:

Block Type	Block Status	Transaction	Reward
regular+	on peer chain	Valid	r_n + `Tx` fee
regular+	on peer chain	Invalid	r_n
regular+	forked from peer chain	Valid	0
regular+	forked from peer chain	Invalid	0
milestone	on the longest milestone chain	Valid	r_m + `Tx` fee + $\delta R(\mathtt{B}_m)$
milestone	on the longest milestone chain	Invalid	$r_m + \delta R(\mathtt{B}_m)$

Table 1: Reward Scheme

On top of fixed amount r_m and the transaction fee if valid, a milestone block also brings additional reward of $\delta R(\mathtt{B}_m)$. We can think of the parameter δ as for example 2%. This means that any miner who created a milestone, he will get in addition 2% of the rewards for all the blocks that milestone confirmed in its level set. For example, if a system is running at 1000 TPS, and milestones are generated every 10 seconds, there will be on average 10000 blocks in a level set. So there is quite a big bonus compared with the situation where the block, despite qualified to be a milestone, end up being a regular+ block because it is not on the longest chain. By appropriately choice of the parameters r_n, r_m and δ, we hope to provide miners enough incentive to try their best to point to the most recent milestone when creating new blocks. Since the extra bonus depend on the number of blocks in the level set, this will also incentivize a miner to point a new block to a recent regular block on another peer chain to maximize the number of blocks he can confirm.

The economic model is out of the scope of this paper which focus on consensus. Under our design, one can choose to adjust r_n and r_m every once in a while so that the total number of issued coins is fixed like in Bitcoin. Alternatively, one can keep them fixed so that the total amount of currency in the system will inflate. There are many ways to design the financial system, which we will not discuss here. No matter how to choose these parameters, such a scheme together with smaller blocks which can be miner more easily will make miner i's income rate commensurate (without too much volatility) to his q_i proportion of total hashing power times the rate at which new currencies are issued plus transaction fees.

Note that in the above table, forked blocks from a peer chain brings zero reward thus incentivize miners to follow the principle of maintaining the structure in peer chains. How exactly does this work will be discussed in the following part.

Registration and Redemption. As we can see in the designed scheme, the reward for a block cannot be recorded as a static numerical number upon the time creating the block. Thus, we need a mechanism that rewards each miner *after* consensus has been reached on the DAG. One possible solution is to build a script in each block which will be able to calculate the exact reward based on the three factors in the table above. But this solution costs too much storage overhead for a block with only one transaction since a script itself will contain address and public key accumulating to about the similar size as a transaction. It is also computationally expensive since we find the `secp256k1 ECDSA` key verification time consuming. We hence propose the following registration-redemption solution The first block each miner creates (the one directly points to the genesis) should contain a special transaction called *registration*, which specifies an address where future rewards on the peer chain should be awarded to. Every once in a while, a miner can choose to create a block on his peer chain contain another special transaction called *redemption*, which plays two roles. The first is to redeem the accumulated reward since the previous redemption (registration) transaction on his peer chain to the address specified by the previous redemption (registration) transaction. The second is to specify the address to which next redemption shall be awarded to. To publish a redemption block, a miner has to prove the ownership of the address in his previous redemption block by using his secrete key. As such, the reward of a miner can only be claim by himself. Also, a miner can choose to redeem whenever he wants and have the option to specify a different address to store his reward every time he redeems. Since a redemption block can only redeem all the rewards in the blocks on the peer chain since the previous redemption block, this effectively prevent a miner from forking his own peer chain since in that way, the forked block costs computing power but brings no reward.

This design also prevents the forking attack on a peer chain by another miner. Suppose a malicious miner, Alice, try to fork the peer chain of an honest miner Bob. Alice first spends some hashing power to create a block whose `peer` is set to Bob and pointer $\texttt{id}_{\texttt{prev}}$ is set to point to some block on Bob's peer chain. If the chosen block is not the head of Bob's peer chain, Alice creates a fork on Bob's peer chain. Alice can surely extend this fork by continuing spending mining power to create more blocks along it. Note that Alice cannot publish a redemption block on Bob's chain since that requires Bob's private key to prove the ownership of the address in

Bob's previous redemption block. Thus there is no way Alice can redeem any reward on Bob's chain, even the reward in the forked blocks which are created by Alice. In fact, this actually gives Bob a choice if the rewards along the fork is larger, Bob can easily append a redemption block to that fork to redeem the rewards. Of course, Bob has to give up the blocks mined by himself by doing so. Such an attack causes no damage to consensus and Bob's revenue while Alice's hashing power will be wasted.

In summary, the registration and redemption approach help to determine a unique peer chain for each miner by embedding a sequence of redemptions block which requires that miner's signature. Note that a simple idea is to insert a signature in every block in order to determine a unique peer chain. But a signature require substantial space in the block, causing too much storage and computation (for signature verification) overhead.

Depth-first Search. The final preparation before constructing the ledger is to provide an ordering of all the transactions (equivalently blocks). Recall in Section 3 that blocks are organized in level sets $\mathcal{S}(1)$, $\mathcal{S}(2)$, $\mathcal{S}(3)$, ... along the longest milestone chain. So we order the level sets using the heights of their milestones from low to high. According to the definition (11), level set $\mathcal{S}(n)$ is a directed binary tree with the root being the nth milestone $\mathtt{B}_{m,n}$. Each directed edge in the tree is essentially a pointer in $\{\mathtt{id}_{\mathtt{prev}}, \mathtt{id}_{\mathtt{tip}}\}$. Note that we remove $\mathtt{id}_{\mathtt{ms}}$ since it either points to a previous level set or is redundant. We use the depth-first search (DFS) to traverse this tree starting from the root $\mathtt{B}_{m,n}$ along the directed edges in the order of first $\mathtt{id}_{\mathtt{prev}}$ and then $\mathtt{id}_{\mathtt{tip}}$. Blocks in each level set are ordered according the post-order DFS.

Assume we obtain an ordered list of transactions following the above level set ordering and DFS ordering within each level set,

$$\mathtt{Tx}_1, \mathtt{Tx}_2, \mathtt{Tx}_3, \ldots$$

Let $\mathcal{L}_k$ be the ledger constructed based on the first k transactions and $\mathcal{U}_k$ denote the set of UTXO based on $\mathcal{L}_k$. We start with $\mathcal{L}_1 = \{\mathtt{Tx}_1\}$ and $\mathcal{U}_1$ being the outputs of $\mathtt{Tx}_1$. In order to obtain $(\mathcal{L}_{k+1}, \mathcal{U}_{k+1})$ from $(\mathcal{L}_k, \mathcal{U}_k)$ and $\mathtt{Tx}_{k+1}$, we need to perform the following check. If the inputs of $\mathtt{Tx}_{k+1}$ are all from $\mathcal{U}_k$ and $\mathtt{Tx}_{k+1}$ passes signature verification, then

$$\begin{aligned}
\mathcal{L}_{k+1} &= \mathcal{L}_{k+1} \cup (\mathtt{Tx}_{k+1}), \\
\mathcal{U}_{k+1} &= \mathcal{U}_k \cup \{\text{outputs from } \mathtt{Tx}_{k+1}\} \setminus \{\text{inputs from } \mathtt{Tx}_{k+1}\},
\end{aligned}$$

otherwise $(\mathcal{L}_{k+1}, \mathcal{U}_{k+1}) = (\mathcal{L}_k, \mathcal{U}_k)$.

We would like to point out that any graph traversal algorithm leading to a unique ordering will be good enough for consensus purpose but an advantage of the DFS ordering is that the chronological order of blocks on a peer chain is kept.

6 Performance Analysis and Evaluation

6.1 Modelling Analysis

We first provide a modeling approach to analyze the performance of the system. We focus on the three most important measures—consensus, TPS, and latency. Throughout our analysis, we make the following synchronization assumption.

Broadcast Delay Assumption. When a peer broadcasts a message of size ν size to a peer-to-peer network with n peers, $F(t)$ fraction of the peers will receive the message after t amount of time. Clearly, $F(t)$ increases with time t. We further assume there exists a finite time t_0 such that $F(t_0) = 1$. In other words, all peers will be able to receive the message within t_0 amount of time.

Note that the broadcast curve F and the bound t_0 depend on both the message size ν and the number of peers n. We point out that the block size in our DAG (typically less than one kilobyte in our implementation) is much smaller than that of Bitcoin for example, which is one megabyte. So our t_0 should be quite small. By classical results in regular graphs, a block will be able to reach all n honest peers in $O(\ln(n))$ relays in expectation, assuming it is syntactically valid such that all honest peers will relay the block immediately after receiving it.

6.1.1 Reaching Consensus

Since all blocks will be confirmed by some milestone block along the longest milestone chain, our level sets essentially play the role of the blocks in Bitcoin. So the consensus of our DAG system essentially boils down to that of the block chain. This has been formally proven by [5] who proposed the Bitcoin Backbone Protocol model. We will not repeat the proof here. Instead, we make the connection to convince readers that it is legitimate for us to directly borrow their result with the following assumptions.

The analysis in [5] is based on discrete rounds, which is similar to our model. We can discretize our time line into intervals of length t_0 and let time interval $(rt_0, (r+1)t_0]$ be round r. Based on our broadcast delay assumption, blocks produced in round r will reach all honest peers by the end of round $r+1$. In the round-based model, we assume all actions are taken at the end of each round.

Let f be the probability that there exists an honest peer producing one milestone in a round. The *honest majority* assumption requires that the number of malicious peers κ, constitute only a small fraction of the total population. Specifically, the number of malicious miner

$$\kappa \leq (1-2f)n$$

Note that a large f intuitively implies a high chance that honest miners will create more than one milestones miners in the same round. This would increase the chance of their forking the milestone chain due to broadcast delay. Thus the larger the f, the smaller the proportion of

malicious miners required for the following result to hold true. In fact, more subtle relationships exist between f and (n, κ). Interested readers can refer to the proofs in [5] for more details. The good news is that under the honest majority assumption, we have the following result on consensus. The following property follows from Theorem 15 in [5].

Common Prefix. Consider any pair of honest peers p_1 and p_2 following our protocol to maintain their local DAGs. Let $\mathcal{G}_{p_1}$ be the local DAG of peer p_1 at round r_1, and $\mathcal{G}_{p_2}$ be the local DAG of peer p_2 at round r_2. Suppose $r_1 \leq r_2$ and the height of $\mathcal{G}_{p_1}$ is h. For any positive integer h', let $\mathtt{B}_{m,h'}$ be the milestone of height h' in $\mathcal{G}_{p_1}$, and $\mathcal{C}_{p_1}(\mathtt{B}_{m,h'})$ be the DAG confirmed by $\mathtt{B}_{m,h'}$ in p_1's local DAG. There exists a k such that $\mathcal{C}_{p_1}(\mathtt{B}_{m,h-k}) \subseteq \mathcal{G}_{p_2}$ with probability higher than $1 - e^{-\Omega(k/f)}$, where $\Omega(x)$ dominates x as x grows.

Note that the Bitcoin Backbone Protocol model is well designed for proving the consensus property, as we are only interested in the existence of the number k and the probability $1 - e^{-\Omega(k/f)}$ in the above. However, it is not suitable for parameter selection. For example, to ensure the probability $1 - e^{-\Omega(k/f)}$ exceeds 99.9%, we may end up with a too large a k for practical use. This is because the estimations in the backbone model are quite conservative.

We propose a more accurate model in the next subsection for setting a practical parameter k.

6.1.2 TPS and Wasted Capacity

Ideally, our design can use up network broadcast capacity, the ceiling of TPS. However, due to synchronization issues, a transaction may be processed by multiple peers and put in several blocks. The extra copies are a waste of broadcast network capacity and hashing power. In this subsection, we provide an upper bound for the wasted capacity under our transaction assignment protocol (12).

In the subsequent analysis, we assume there are n honest miners with equal hashing power. We also assume that with the current total hashing power, new blocks are created following a Poisson process with rate $n\mu$ (blocks/unit of time) which can be viewed as the highest rate that the network allows peers to broadcast blocks, where μ denotes the average rate of block creation for each miner.

Suppose that transaction $\mathtt{Tx}$ reaches miner i at time 0. However, $\mathtt{Tx}$ has already been mined by some miner, who has subsequently broadcasted his block $\mathtt{B}_{\mathtt{Tx}}$ containing this transaction to the network at time 0. Denote by T_i the amount of time for miner i to receive this block. According to our broadcast delay assumption, T_i is a random variable following distribution F. Let us assume the worst case where the transaction fee of $\mathtt{Tx}$ is so attractive that miner i will surely mine this transaction whenever the it is workable at a time before T_i. Let $N_i(t)$ denote the number of blocks miner i can create during time $(0, t]$. Then, $N_i(\cdot)$ is a Poisson process with rate μ. During $(0, t]$, miner i changes his peer chain head $N_i(t)$ times, which follows a Poisson distribution with rate μt. Thus, conditional on $N_i(t) = k$, the probability that $\mathtt{Tx}$ is workable

for miner i some time in $(0, t]$ is

$$\mathbb{P}(I_i(t) = 1 | N_i(t) = k) = 1 - (1 - \frac{c}{n})^k \leq 1 - e^{-\frac{ck}{n}},$$

where $I_i(t)$ indicates whether or not miner i is eligible to work on `Tx`. Let A_i denote the event where miner i successfully mines `Tx` before he receives the block by time T_i. We have the conditional probabilities

$$\begin{aligned}
&\mathbb{P}(A_i | I_i(T_i) = 1, T_i = t) \leq 1 - e^{-\mu t}, \\
&\mathbb{P}(A_i | I_i(T_i) = 0, T_i = t) = 0
\end{aligned}$$

where $1 - e^{-\mu t}$ is the probability that miner i will successfully create the block in time t and this is an upper bound for the left side of the first expression. Thus, conditional on $T_i = t$, the probability that miner i creates a block for transaction `Tx` can be bounded as

$$\begin{aligned}
\mathbb{P}(A_i | T_i = t) &\leq (1 - e^{-\mu t}) \mathbb{P}(I_i(T_i) = 1 | T_i = t) \\
&= (1 - e^{-\mu t}) \mathbb{E}_{N_i}[1 - (1 - \frac{c}{n})^{N_i} | T_i = t] \\
&= (1 - e^{-\mu t})(1 - e^{-\mu t \frac{c}{n}})
\end{aligned}$$

This implies

$$\mathbb{P}(A_i) \leq \mathbb{E}(1 - e^{-\mu T_i})(1 - e^{-\frac{c}{n} \mu T_i}) \leq (1 - e^{-\mu \bar{t}})(1 - e^{-\mu \bar{t} \frac{c}{n}}),$$

where $\bar{t} = \mathbb{E}(T_i) = \int_0^{t_0} t dF(t)$, and the last inequality is obtained by Jenson's inequality. The probability that `Tx` is mined exactly once is lower-bounded by

$$\prod_{i=1}^{n} (1 - \mathbb{P}(A_i)) \geq (e^{-\frac{c}{n} \mu \bar{t}}(1 - e^{-\mu \bar{t}}) + e^{-\mu \bar{t}})^n \to e^{-c \mu \bar{t}(1 - e^{-\mu \bar{t}})}$$

as $n \to \infty$. The expected number of copies of mined `Tx` is upper-bounded by

$$1 + \sum_{i=1}^{n} \mathbb{P}(A_i) \leq 1 + (1 - e^{-\mu \bar{t}}) n (1 - e^{-\mu \bar{t} \frac{c}{n}}) \to 1 + (1 - e^{-\mu \bar{t}}) \mu c \bar{t}$$

as $n \to \infty$. So the proportion of capacity that is wasted is upper-bounded by

$$\theta(c) = \frac{(1 - e^{-\mu \bar{t}}) \mu c \bar{t}}{1 + (1 - e^{-\mu \bar{t}}) \mu c \bar{t}}. \tag{13}$$

where c is a design parameter. A smaller c results in a lower level of capacity waste.

6.1.3 Latency

After a transaction enters the mempool, it will go through three phases before it is finally confirmed in the public ledger. Firstly, the transaction has to wait in the mempool until some miner creates a block `B` to store it. We call this waiting time the *queueing latency* W_1. Next, this block needs to be confirmed by a milestone $\mathtt{B}_m$. Recall the definition of confirmation given in (10). We call this period the *infection latency* W_2 since we will analyze it through an infection model. Lastly, the milestone $\mathtt{B}_m$ needs to be extended by a number of future milestones on the longest chain to ensure a certain level of security. We call the last stage *secure latency* W_3.

Queueing Latency. Suppose new transactions arrive at the mempool following a counting process with a constant rate λ. The mempool is essentially a queueing system with arrival rate λ and effective processing rate depending on both $n\mu$ and the transaction assignment rule (12). We now give an estimation of the waiting time in queue W_1 based on a fluid idea in queueing theory.

Denote by Q the stable queue length, i.e. number of transactions in the mempool. Whenever a miner tries to find a transaction from the mempool to work on, he will find that the number of transactions he can process follows the binomial distribution with total number of trials Q and success probability c/n, which can be approximated by a Poisson random variable with rate (Qc/n) since Q is large and c/n is small. So the proportion of time that a miner has to work on an empty block is the probability that the Poisson random variable equals 0, i.e. $e^{-Qc/n}$. The rate at which blocks containing transactions are generated is therefore $n\mu(1-e^{-Qc/n})$. As noted in the previous section, only $(1-\theta(c))$ of the transactions are distinct, so the rate at which the transactions in the mempool are processed is $(1-\theta(c))n\mu(1-e^{-Qc/n})$. For the system to have a stable Q, it is required that

$$n\mu(1-e^{-Qc/n})(1-\theta(c)) = \lambda.$$

Solving the above equation yields $Q = \frac{n}{c}\ln(\frac{n\mu}{n\mu-\lambda/(1-\theta(c))})$. By Little's law, the average waiting time of a transaction is

$$W_1 = \frac{n}{c\lambda}\ln\left(\frac{n\mu}{n\mu-\frac{\lambda}{1-\theta(c)}}\right) = \frac{1}{c}\frac{1}{\rho\mu}\ln\left(\frac{1}{1-\frac{\rho}{1-\theta(c)}}\right), \tag{14}$$

where $\rho = \frac{\lambda}{n\mu}$ denotes the traffic intensity. Note that the number of miners does not affect the queueing latency as long as the rate at which new blocks are produced remains constant. The influence of c on W_1 is complicated. On the one hand, a larger c will result in less idle time, thus increasing the effective processing rate. On the other hand, a larger c leads to a higher proportion of duplicate blocks, thus decreasing the effective processing rate. Eliminating c from (13) and (14), we can describe the relation between wasted capacity and queueing latency follows:

$$W_1(\theta) = \frac{(1-\theta)\bar{t}(1-e^{-\mu\bar{t}})}{\theta\rho}\ln\left(\frac{1}{1-\frac{\rho}{1-\theta}}\right) \tag{15}$$

Our quantitative modeling analysis sheds the light on how the parameter c can be chosen to strike a balance between collision and latency.

Traditional queueing theory suggests that the waiting time will blow up when the traffic intensity $\frac{\rho}{1-\theta}$ approaches 1. A more complicated model can be analyzed by allowing each transaction to have an expiration clock, without which the mempool size will grow without a bound. For the time being, the above queueing model is convincing enough to get the system started.

Infection Latency. A primary difference between the DAG and chain structures is that the former goes beyond the one-dimensional linear structure and allows parallelism. Despite the

advantages of the DAG, one issue is that a block may not necessarily be confirmed by the next milestone—it may have to wait for a later milestone.We now provide an upper bound on this additional waiting time by modeling confirmation in our DAG using an infection model. In our analysis, we assume all of the n miners are incentivized to follow the three principles specified in Section 5.

Suppose that at time 0, block B is in the pending set, i.e. it has not been confirmed by any milestone. As previously discussed, new blocks arrive following a Poisson process with rate $n\mu$. Each new block is a milestone with probability p. Suppose that at time $s > 0$, there are X_s miners whose head block can reach B through a directed path in the DAG. We say that these miners are *infected* by B. Note that if all miners are infected, then B will surely be confirmed by the next milestone. Assume the next new block is created by miner a at time $t > s$. Assuming he randomly picks any of the n chain heads, he will become infected with probability $\frac{X_s}{n}$. Note that this probability is lower than the actual probability hat B is confirmed by t, because block B may have already been confirmed by a milestone if the chain head of any of the X_s infected miners's is a milestone. Since we are considering an upper bound, we may assume $X_0 = 1$ and

$$X_t = \begin{cases} X_s + 1 & \text{with probability } \frac{X_s(n-X_s)}{n^2}, \\ X_s & \text{with probability } 1 - \frac{X_s(n-X_s)}{n^2}. \end{cases} \tag{16}$$

We also introduce M_s, which indicates whether or not B is confirmed by a milestone. So $M_0 = 0$ and

$$M_t = \begin{cases} 1 & \text{with probability } p(\frac{X_s}{n} + \frac{X_s(n-X_x)}{n^2}), \\ 0 & \text{with probability } 1 - p(\frac{X_s}{n} + \frac{X_s(n-X_x)}{n^2}). \end{cases} \tag{17}$$

The above modeling gives a continuous-time Markov chain (X_t, M_t) and we are interested in the expected time taken to hit the set $\{(x, m) : x \geq 1, m = 1\}$. Let q_x denote the expected jumps needed to hit the set starting from $(x, 0)$. Then

$$\begin{aligned} q_x &= 1 + q_{x+1}\frac{x(n-x)}{n^2}(1-p) + q_x(\frac{x}{n}(1-p) + \frac{(n-x)^2}{n^2}) \\ q_n &= \frac{1}{p}. \end{aligned}$$

Therefore,

$$\begin{aligned} q_1 &= \sum_{k=1}^{n} \frac{n^2}{nj(1+p) - j^2} \prod_{j=1}^{k-1} \frac{(1-p)(n-k)}{n(1+p) - k} \\ &< 2n(1 + \ln(n)) + \frac{1}{p}. \end{aligned}$$

Let $\tau_1 := \inf\{t : M_t = 1\}$, the time needed for B to be confirmed by a milestone. Since blocks arrive at the rate $n\mu$, which is exactly the rate for all jumps, the infection latency is

$$W_2 = \mathbb{E}(\tau) = \frac{1}{n\mu} q_1 < \frac{2 + 2\ln(n)}{\mu} + \frac{1}{np\mu}. \tag{18}$$

Note that the second term in the above upper bound is basically the expected time it takes for a milestone to arrive, which is fixed. Another contribution to the infection latency comes from

$\ln(n)$ in the first term. Intuitively, the infection latency grows with the number of miners, n, due to parallelism. However, the growth is of logarithmic rate and thus, the infection latency will not be large even there is a large number of miners.

Secure Latency. After a block is confirmed by a milestone, we still need to wait for this milestone to be extended by a number of future milestones for security guarantee. This is essentially the same latency that occurs in blockchain systems such as Bitcoin. This type of latency was analyzed in [9] in a simple model assuming honest miners will not fork among themselves. However, honest miners may fork among themselves due to broadcast delay because when an honest miner creates a new block, he may not be aware of a recent block created by another honest miner. A round-based model was formulated by [5] to handle this situation. The synchronization assumption is that whatever happened in the previous round will be made known to all honest miners in the present round so that they can act accordingly. However, they analyse their model in a worst-case scenario and is too conservative for parameter selection. We now describe a continuous-time model to incorporate the broadcast delay function F and the potential forking among honest miners.

Consider the arrival process of milestones created by honest peers. Let us call such milestones *honest* milestone blocks. The creation of honest milestone blocks is a Poisson process with rate $pn\mu$. Let $U_1, U_2, U_3, \ldots$ denote the inter-arrival times of milestones in this process. They are independent and follow the exponential distribution with rate $pn\mu$. Let us call the ith arrival *milestone* i. This milestone is chronologically the ith milestone created by the honest peers, but it may not be the ith milestone ever created in the network due to the existence of malicious miners. We would like to tag milestone i with a value $Y_i \in \{0, 1\}$, for $i = 1, 2, \ldots$, in such a way that if $Y_i = 1$ then the creator of milestone i will have received all preceding milestones created by all honest miners upon the creation of milestone i. The tag $Y_i = 1$ implies that milestone i must be higher than any previous milestone tagged with a 1 because the creator of milestone i is aware of all previous type-1 milestones when he is creating milestone i. Intuitively, being tagged with a 1 is a good signal and will likely lead to a height increment of the longest chain. It follows from our tagging method that, among all milestones of the same height in the milestone tree, at most one can be tagged with a 1 as illustrated in Figure 4, where * represents a milestone of type-0 or a milestone created by a malicious peer.

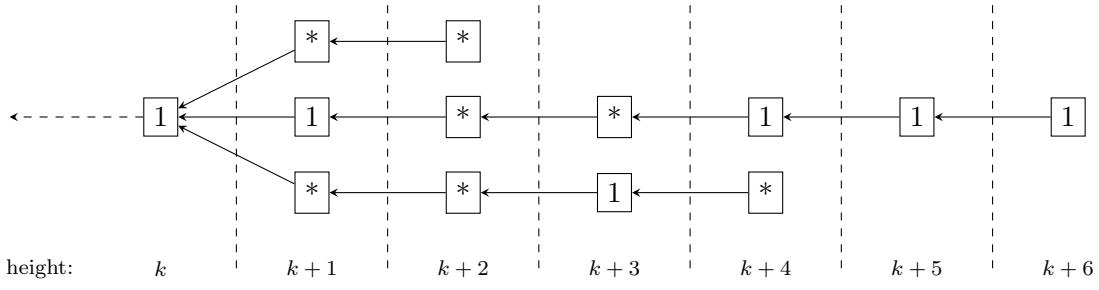

Figure 4: Illustration of Tags in the Milestone Tree

The first arrival is tagged with a 1, i.e., $Y_1 = 1$, as it is the first milestone created by honest peers after the genesis `0`. Let us now think about the circumstances in which a future honest milestone, say milestone i, may be tagged with a 0. Consider the case where its preceding honest milestone is tagged with a 1, i.e., $Y_{i-1} = 1$. If the inter-arrival time between milestone $(i-1)$ and milestone i, U_i, exceeds t_0, then the honest miner who creates milestone i will have certainly received all honest milestones. Suppose $U_i = t \in (0, t_0)$, then according to the broadcast delay assumption, the honest miner who creates milestone i have probability $F(t)$ of having received milestone $(i-1)$ and thus all preceding honest milestones. So milestone i will be tagged with a 1 with probability $F(t)$. In general, let Z_i be a Bernoulli random variable with success probability

$$\mathbb{P}(Z_i = 1) = \int_0^{t_0} F(t) pn\mu e^{-pn\mu t} dt + e^{-pn\mu t_0}.$$

Milestone i will be tagged with a 1 if $Y_{i-1} = 1$ and $Z_i = 0$. Now consider the case where the preceding honest milestone is tagged with a 0, i.e., $Y_{i-1} = 0$. We would like to be conservative by tagging milestone i with a 0 whenever $U_i < t_0$. In this case, only when the inter-arrival time U_i exceeds t_0 can we be sure that the honest miner who creates milestone i has received all preceding honest milestones and we will then tag milestone i with a 1. Mathematically,

$$Y_i = \begin{cases} 0, \text{ if } (Y_{i-1} = 0 \text{ and } U_i \le t_0) \text{ or } (Y_{i-1} = 1 \text{ and } Z_i = 0), \\ 1, \text{ otherwise.} \end{cases} \tag{19}$$

The milestone chain evolves as follows. First, a number of milestones are tagged with a 1 meaning the longest milestone chain will grow whenever an honest peer produces a milestone. When the first type-0 milestone arrives, it may be of the same height as a previous type-1 milestone, and thus it could potentially lead to forks. Once a type-0 milestone arrives, all newly arriving milestone blocks will be regarded as useless until another milestone tagged with a 1 arrives, the height of which will exceed that of any milestone previously mined by honest peers. Beyond that point in time, the regenerative cycle restarts. This is a conservative model, because after a type-0 milestone arrives, the miner of some subsequent milestone with inter-arrival time less than t_0 may be informed of all preceding honest milestones if he is lucky enough, but an inter-arrival time greater than t_0 will ensure that he will be informed.

In each regenerative cycle, the milestones tagged with a 0 can be regarded as wasted. Actually, we can consider the wasted milestone as if they were created by malicious miners. In other words, the effective hashing power of honest miners should be discounted by the proportion of the blocks tagged with a 1 in a cycle. In each cycle, the number of blocks tagged with a 1 follows the geometric distribution with success probability $\int_0^{t_0}(1-F(t))pn\mu e^{-pn\mu t}dt$ and the number of blocks tagged with a 0 is geometric with success probability $e^{-pn\mu t_0}$. Thus, the long-run average proportion of milestones tagged with a 1 can be estimated as

$$\frac{e^{-pn\mu t_0}}{e^{-pn\mu t_0} + \int_0^{t_0} pn\mu(1-F(t))e^{-pn\mu t}dt}.$$

For instance, if $pn\mu = 0.1/s$, $t_0 = 2s$ and $F(t) = t - t^2/4$, then the above fraction will be 0.928. So in a network where 10% of miners are malicious and 90% are honest, at least $90 \times 0.928\%$

of the hashing power would be devoted to growing the chain. Thus, one method to compute the number of milestones we need to wait for is to simply replace 90% with $90 \times 0.928\%$ in the Nakamoto model [9]. We also consider perform simulation to guide the selection of secure latency.

6.2 Numerical Studies

Now we provide numerical results from simulations to evaluate the performance of our design. Throughout the discussion, unless specified otherwise, we consider the scenario with $n = 1200$ miners, individual block production rate $\mu = 1$ per second, milestone proportion $p = 1/12000$, and transaction arrival rate $\lambda = 1000$ per second. Thus, milestones are created at a rate of $pn\mu = 1$ per 10 seconds.

Wasted Capacity vs. Queueing Latency As mentioned in our modelling analysis, there is a trade-off between wasted capacity and queueing latency through the design parameter c. To simplify the simulation, we study a more conservative case. That is, we assume that once a miner selects a workable transaction from the mempool, he will mine a block to contain that transaction for sure, while in reality, a miner may receive a block that contains the transaction that he is currently working on, in which case he may switch to another transaction, reducing the wasted capacity. Figure 5 illustrates the trade-off between wasted capacity and queueing latency as we vary c from 1 for 10^{-4}. As we desire both a short queueing latency and a low proportion of wasted capacity, we may select as a point on the curve that is close to zero. For instance, the wasted capacity is around 5% when W_1 is around 13 seconds, in which case the design parameter c is set as 0.1.

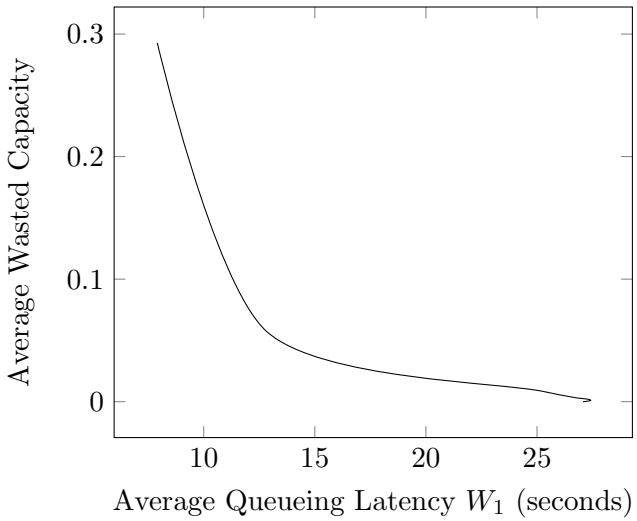

Figure 5: Wasted Capacity vs. Queueing Latency

Infection Latency The infection latency depends mainly on the number of miners n and individual mining rate μ. From our simulation results, the infection latency W_2 is around 17

seconds in the scenario under consideration.

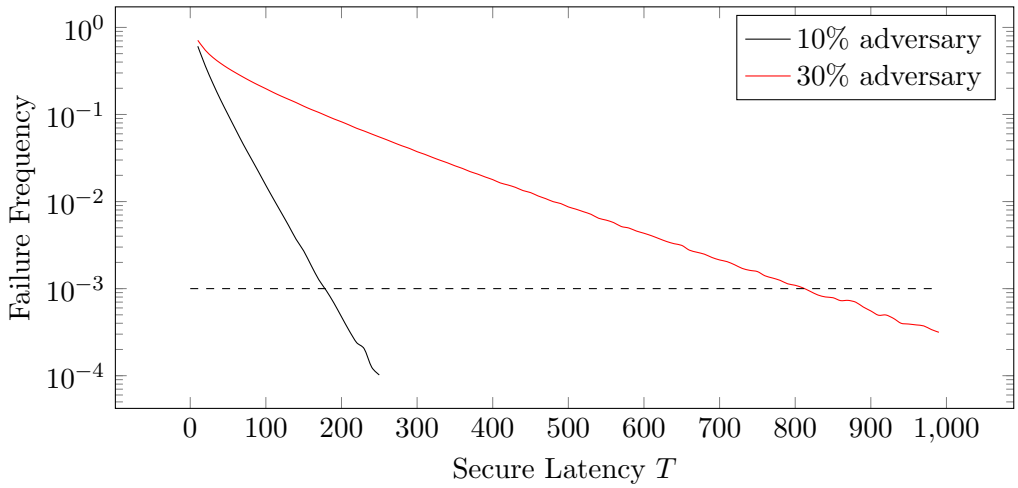

Figure 6: Failure Frequency based on Simulation of 10^6 Sample Paths

Secure Latency We perform simulations based on our tagged-arrival model to determine how long the secure latency needs to be in order to ensure a given level of security. Suppose a milestone $\mathtt{B}_m$ is created at time t, and is still part of the longest chain in the local DAG of some honest peer after a period of time T. We want $\mathtt{B}_m$ to be in the longest milestone chain from every honest peer's perspective and to stay in the longest chain thereafter with a high probability. Suppose at time $t+T$, milestone $\mathtt{B}_m$ is on the currently longest chain C_1 of one honest peer, and there exists another honest peer adopting a chain C_2 which does not contain $\mathtt{B}_m$. Consider the type-1 arrivals during $[t+t_0, t+T-t_0]$, all of them are higher than $\mathtt{B}_m$ and are received by all honest peers by $t+T$. Suppose that $\mathtt{B}$ is one of them and is of height $h > \eta(\mathtt{B}_m)$. Both C_1 and C_2 have a block of height h, say $\mathtt{B}_1$ and $\mathtt{B}_2$ respectively, since both peers have received $\mathtt{B}$ and C_1 and C_2 are the longest chain from each peer's own point of view. In addition $\mathtt{B}_1 \neq \mathtt{B}_2$ as both of them are higher than $\mathtt{B}_m$ but only $\mathtt{B}_1$ extends $\mathtt{B}_m$. Hence either $\mathtt{B} \neq \mathtt{B}_1$ or $\mathtt{B} \neq \mathtt{B}_2$ or both are true. Therefore the number of 1s during $[t+t_0, t+T-t_0]$ is less than the total number of 0s plus the number of milestones created by malicious miners during $[t, t+T]$. The probability of this event is low when T is sufficiently large and decays rapidly with respect to T as shown in our simulation results in Figure 6. To be on the conservative side, we start our simulation with the first tag being 0, which is stochastically a worse initial condition. It can be seen from the figure that we have to wait 130 seconds and 810 seconds assuming 10% and 30% of the hashing power comes from malicious miners, respectively, to ensure a failure frequency of less than 10^{-3}. We use $pn\mu = 0.1/s$, $t_0 = 2s$ and $F(t) = t - t^2/4$ in our simulation. This means once a transaction is first confirmed by a milestone, we need to wait for another 81 and 13 milestones on average before finally accepting the milestone assuming 10% and 30% of the hashing power are attributed to malicious miners, respectively.

Discussion on Parameter Selection We now summarize the performance of our suggested parameters. Suppose that $n = 1000$ honest miners in the system, each creating blocks at the rate μ of 1.2 blocks per second. We also assume that the mempool assignment parameter $c = 0.1$ in (12). So the max TPS of the design is approximately $95\%n\mu = 1140$ according to our simulation result. The total latency is $W = W_1 + W_2 + W_3$, where W_1 is the queueing latency, W_2 is the infection latency and W_3 is the secure latency. By assuming that transactions arrive at the rate λ of 1000 per second, the queueing latency W_1 is approximately 13 seconds according to (14). If we further assume that the expected inter-arrival time of milestones is $1/pn\mu = 10$ seconds (i.e., $p = 1/12000$), the infection latency W_2 is approximately 17 seconds. With the broadcast curve $F(t) = t - t^2/4$ as before, $t_0 = 2$ and $\bar{t} = 5/3$. Suppose also that less than 30% of hashing power comes from malicious miners. If we want to ensure that the probability of a successful attack is less than 10^{-3}, then W_3 is 810 seconds according to the simulation shown in Figure 6. So the total latency W is approximately 840 seconds or 14 minutes.

7 Conclusion and Discussion

In this paper, we design an efficient consensus mechanism based on the novel structured DAG and a protocol for building the public ledger from the DAG. Our designs shorten the latency for reaching consensus, increase the TPS, and discourage the need for the large mining pool. We also develop a transaction assignment mechanism based on the DAG to reduce the probability of collision, and hence decrease the waste of capacity. Furthermore, we provide both theoretical and numerical guarantees for the performance of our designs. Many other problems still exist in the emerging field of blockchain and cryptocurrency. For example, as TPS increases to the thousands, a dozen terabytes of data will easily accumulate each year. The current reward scheme only incentivizes people to provide mining services to a public ledger. How do we design an economic model that incentivizes people to provide storage and the required bandwidth? We will leave this kind of problem for future research.

Response to Reviewer 1 We thank the referee for the very careful review and help comments. We have addressed all your comments in the following point-to-point response.

1. In abstract, "render the negligible probability that a transaction is being processed by more than one miners" rather than "render negligible the probability that a transaction being processed by more than one miners"?

   Reply We rewrite that sentence as 'The method reduces the probability that a transaction is processed by multiple miners and hence improves processing efficiency.'

2. The storyline in abstract is incomplete. In particular, the background and motivation are unclear.

   Reply We rewrite the abstract and make the background and motivation clear.

3. The introduction is too long. The description of the proposed consensus mechanism is wordy and should be simplified.

   Reply We modify our introduction and simplify the description of consensus.

4. The authors should summarize the main contributions at the end of introduction.

   Reply We summarize our main contributions at the last two paragraphs in introduction.

5. Some important sections, i.e., related works and conclusion, are missing as in the manuscript.

   Reply We add two sections, Literature review and Conclusion.

6. IOTA is not the only project employing DAG consensus, some other projects include but are not limited to Byteball and Nano. In academia, there are also some important works about DAG consensus, e.g., "LDV: A Lightweight DAG-Based Blockchain for Vehicular Social Networks", "CoDAG: An Efficient and Compacted DAG-Based Blockchain Protocol", and "Blockchain Meets DAG: A BlockDAG Consensus Mechanism". Some other important works include "blochie: a blockchain-based platform for healthcare information exchange" and "privacy-preserving and efficient multi-keyword search over encrypted data on blockchain". Also, note that the result data structure of sharding protocols is also DAG.

   Reply We mention these papers in Literature review.

Response to Reviewer 2 We thank the referee for the very careful review and help comments. We have addressed all your comments in the following point-to-point response.

1. The content of the abstract is too simple and lacks background and motivation.

   Reply We rewrite the abstract, articulating the goal to improve capacity as it prevents a wider application of blockchain technology and is a fundamental challenge.

2. There is only theoretical analysis while lacking real-world experiments. Since blockchain is an application-level technology, prototype and real-world experiments are important.

   Reply In addition to theoretical analysis, we report simulation-based experiment methods and results in Section 6.2.

3. Lack of discussion and conclusions.

   Reply We provide conclusions on our contribution as well as discussion on potential future researches in Section 7.

4. Some format and writing standard issues need attention

   Reply Thank you, we have modified the footnotes and notations according to your suggestion. We also reorganize Section 3 (Section 2 in the previous version)in a more formal way.

References

[1] Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. Deconstructing the blockchain to approach physical limits. *arXiv preprint arXiv:1810.08092*, 2018.

[2] Jing Chen and Silvio Micali. Algorand. *arXiv preprint arXiv:1607.01341*, 2016.

[3] Laizhong Cui, Shu Yang, Ziteng Chen, Yi Pan, Mingwei Xu, and Ke Xu. An efficient and compacted dag-based blockchain protocol for industrial internet of things. *IEEE Transactions on Industrial Informatics*, 16(6):4134–4145, 2019.

[4] Keke Gai, Ziyue Hu, Liehuang Zhu, Ruili Wang, and Zijian Zhang. Blockchain meets dag: A blockdag consensus mechanism. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 110–125. Springer, 2020.

[5] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology - EUROCRYPT*, pages 281–310, Berlin, Heidelberg, 2015. Springer.

[6] Shan Jiang, Jiannong Cao, Hanqing Wu, Yanni Yang, Mingyu Ma, and Jianfei He. Blochie: a blockchain-based platform for healthcare information exchange. In *2018 ieee international conference on smart computing (smartcomp)*, pages 49–56. IEEE, 2018.

[7] Donald E. Knuth. *The art of computer programming. Vol. 1.* Addison-Wesley, Reading, MA, 1997.

[8] Chenxing Li, Peilun Li, Wei Xu, Fan Long, and Andrew Chi-chih Yao. Scaling nakamoto consensus to thousands of transactions per second. *arXiv preprint arXiv:1805.03870*, 2018.

[9] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008. `https://bitcoin.org/bitcoin.pdf`.

[10] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–33. Springer, 2018.

[11] Serguei Popov. The tangle. *cit. on*, page 131, 2016. `https://iota.org/IOTA_Whitepaper.pdf`.

[12] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 507–527. Springer, 2015.

[13] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 2014. `https://ethereum.github.io/yellowpaper/paper.pdf`.

[14] W. Yang, X. Dai, J. Xiao, and H. Jin. Ldv: A lightweight dag-based blockchain for vehicular social networks. *IEEE Transactions on Vehicular Technology*, 69(6):5749–5759, 2020.