

# CDAG: A Serialized blockDAG for Permissioned Blockchain

Himanshu Gupta

Department of CSE

Indian Institute of Technology Madras

himanshg@cse.iitm.ac.in

Dharanipragada Janakiram

Department of CSE

Indian Institute of Technology Madras

djram@iitm.ac.in

## Abstract

Blockchain is maintained as a global log between a network of nodes and uses cryptographic distributed protocols to synchronize the updates. As adopted by Bitcoin and Ethereum these update operations to the ledger are serialized, and executed in batches. To safeguard the system against the generation of conflicting sets of updates and maintain the consistency of the ledger, the frequency of the updates is controlled, which severely affects the performance of the system.

This paper presents Converging Directed Acyclic Graph (CDAG), as a substitute for the chain and DAG structures used in other blockchain protocols. CDAG allows multiple parallel updates to the ledger and converges them at the next step providing finality to the blocks. It partitions the updates into non-intersecting buckets of transactions to prevent the generation of conflicting blocks and divide the time into slots to provide enough time for them to propagate in the network. Multiple simultaneous updates improve the throughput of CDAG, and the converging step helps to finalize them faster, even in the presence of conflicts. Moreover, CDAG provides a total order among the blocks of the ledger to support smart contracts, unlike some of the other blockDAG protocols.

We evaluate the performance of CDAG on Google Cloud Platform using Google Kubernetes Engine, simulating a real-time network. Experimental results show that CDAG achieves a throughput of more than 2000 transactions per second and confirms them well in under 2 minutes. Also, the protocol scales well in comparison to other permissioned protocols, and the capacity of the network only limits the performance.

**Keywords** CDAG, blockchain, blockDAG, distributed time barrier, bucketing of transactions

## 1 Introduction

Bitcoin, proposed in 2008 by a pseudonym named *Satoshi Nakamoto* [19], introduced the first ever decentralized framework to manage digital currency over a peer-to-peer network. It stores transactions in a publicly available ledger called the *blockchain*. Each block in the blockchain contains a batch of transactions referring back to the latest known block, constructing a chain of blocks. The consistency of the ledger comes from the serial extension, as the newly

added block is legitimate only if it is consistent with the chain. After Bitcoin other cryptocurrencies like Ethereum [33], Litecoin [17] originate extending the functionality of Bitcoin but using the same underlying blockchain structure.

To maintain the consistency of the ledger block generation rate is kept reasonably low because a higher rate can result in forks and increase the risk of a double spend. Also, creating larger blocks with more number of transactions results in higher propagation time, which in turn refrains the system to scale. [9, 26] discuss in detail about the security problems that originate because of high block generation rate or increasing the block size for the blockchain.

BlockDAG [29] is introduced as an alternate structure to address the issue of the scalability of blockchain. It overcomes the problem of low transaction throughput by supporting very high block generation rate and arrange the proposed blocks in a Directed Acyclic Graph instead of a chain. However, the performance still degrades in the presence of conflicting blocks because of their sophisticated transaction confirmation strategies. Also, due to high block generation rates, the probability of concurrent blocks having the same transactions is high. It is indistinguishable from the case of having blocks with double-spend transactions since in both the scenarios, only one of the proposed blocks gets accepted for the final ledger and other needs to be discarded. Thus, the orphan rate for DAG can still have high values, even in the absence of any conflicting transactions in real life deployments.

This paper describes *Converging Directed Acyclic Graph* (CDAG), a new structure to maintain a consistent and distributed ledger over a peer-to-peer network. CDAG is a hybrid structure of blockchain and blockDAG, designed to achieve the best of both worlds. It progresses serially similar to the blockchain and can handle multiple simultaneous blocks analogous to the blockDAG. Splitting the transactions into non-intersecting buckets enable the block proposers to generate multiple non-conflicting blocks simultaneously, and dividing the time into slots with the help of a *Distributed Time Barrier* grants them enough time to disseminate in the network.

Blocks in CDAG converge after each step into a single block known as the *Converging Block* (C-Block) which is the point of reference for the next set of blocks. Adding C-Block at each step enables the ledger to progress as a chain and

maintain a total ordering among the blocks. However, temporary forks can occur because the ledger progresses as a chain. CDAG handles it by using a variant of the heaviest chain protocol [26] to confirm transactions irreversibly in an acceptable amount of time.

The rest of the paper is structured as follows. Section 2 discusses the related work. Section 3 provides a summary of the previous work extended in this paper. Section 4 describes the design of the protocol. Section 5 and 6 discuss the details of CDAG and describe the block proposal mechanism. Section 7 provides a security analysis of the protocol. Section 8 and 9 tells about the implementation and evaluation of the protocol. Finally, we discuss the limitations and indicate future directions in Section 10, and conclude our work in Section 11.

## 2 Related Work

### 2.1 Chains

The chain is a fundamental and most commonly used structure for the blockchain. Blocks of transactions extending the chain are added periodically one after the other, linked cryptographically to their previous blocks. Nodes mine these blocks for the chain to get the associated reward. Bitcoin uses proof-of-work often called as “Nakamoto consensus” to mine blocks, named after the creator of bitcoin. Participants try to solve a hard cryptographic puzzle, and the first one to get through proposes the block. Nodes require massive infrastructure to participate, and the problem is set too hard such that probabilistically only a single node can generate a block every 10 minutes. It incurs a lot of resource wastage, and throughput of the system still suffers. Moreover, due to the possibility of forks, users wait for the chain to grow at least six blocks before considering a transaction confirmed [2], i.e., an hour of waiting. Many follow-up cryptocurrencies like [10, 17, 34] adopted the bitcoin’s proof-of-work and inherited its limitations.

CDAG, on the contrary, allows a set of nodes in each round to propose blocks of transactions simultaneously from distinct buckets. Valid proposals from different proposers are added to the ledger in parallel, extending the ledger as a converging DAG, unlike a chain of blocks used in Bitcoin. It helps CDAG to gain efficiency and increase the throughput of the system.

Bitcoin-NG [11] takes a step forward to use proof-of-work to elect a leader for a time epoch, and let that leader propose blocks until the next leader gets elected. It improves the performance, but a single leader for each epoch is susceptible to an attack. CDAG also progresses in time slots, but there is no unique leader to propose blocks. Multiple proposers simultaneously try to introduce blocks from different buckets. It prevents the system against denial of services (DOS) attacks and ensures the progress of the protocol.

Algorand [13] use anonymous committee selection to propose blocks and perform voting to select the one amongst them to be added to the ledger. It utilizes network bandwidth to propagate  $n$  number of blocks, but finally adds only one amongst them to the ledger. It results in under-utilization of the network resources and compromises on the achievable throughput. CDAG try to maximize the throughput by allowing the blocks from multiple proposers to get added to the ledger concurrently. It tries to exploit the network resources and also gains on throughput.

### 2.2 Trees and DAGs

Other than chains, structures like trees and directed acyclic graph (DAG) have gained eminence lately. GHOST [26] modifies the chain selection rule of Bitcoin to select the heaviest sub-tree instead of longest chain at a fork. It selects the branch with the highest mining power concentration, providing better security guarantees than original bitcoin, with higher block generation rates and improved fairness. CDAG also grows like a tree with multiple child blocks at each step, but they converge at the next step trying to commit a maximum possible of them. The protocol tries to add all the non-conflicting child blocks to the ledger, unlike [26] where only a single child block ends up in the final chain. It results in better performance and supports even higher block generation rates.

Spectre [25] uses directed acyclic graphs as the underlying structure for the ledger instead of chains for increasing Bitcoins throughput. Blocks in [25] points to all the leaf blocks visible to the proposer and resolve conflicts based on the number of direct and derived links for the concerned blocks. It supports a very high block generation rate in comparison to the original bitcoin blockchain and also scales easily without compromising on the security. However, Spectre provides only pairwise ordering between the blocks and not total-ordering which makes it unsuitable for *Smart Contracts* [28] based blockchain deployments like Ethereum. Moreover, it is possible that a block added in the DAG may not ever get confirmed in the presence of conflicting transactions [24]. CDAG, on the other side, converges by design at each step and thus supports smart contracts. Conflicts can occur in the form of forks but are resolved irreversibly as long as a majority of nodes are working towards extending the heaviest branch.

Phantom [27] is also a blockDAG based protocol that provides a total ordering among the blocks added to the ledger. It makes Phantom suitable for smart contracts, but the performance degrades severely in the presence of conflicting blocks. Moreover, in both [25, 27] there is no mechanism for the proposers to generate distinct blocks. Multiple blocks proposed at small intervals possess a high probability of having conflicts, which can result in higher orphan rate (percentage of the proposed blocks not included in the ledger)

in real-time systems. CDAG addresses this issue by dividing the unconfirmed transactions pool into multiple buckets. Multiple proposers randomly select buckets to propose blocks, reducing the probability of the generation of conflicting blocks and therefore decreasing the orphan rate.

### 3 Background

CDAG is a structure to store blocks of transactions proposed for a ledger. It can accommodate multiple blocks proposed simultaneously but requires a distributed consensus protocol to select the proposers for each round. CDAG uses Colosseum to select block proposers for each of the time slot and add the blocks generated by them to the ledger.

#### Colosseum

Colosseum [14] is a scalable consensus protocol to elect a random subset of nodes from a peer-to-peer network. It is designed for permissioned blockchain networks where the identity of participants are known. Colosseum follows a curative approach towards consensus rather than a preventive approach followed by all the voting based algorithms like [6]. It assumes that adversaries are not always present in a network, but efficiently detects them and alerts the network; whereas the traditional Byzantine Fault Tolerant consensus algorithms work on the assumption that traitors are always present in the network. Such a design assumption brings down the message complexity of Colosseum and allows the protocol to scale.

Colosseum arranges the participants in a structured ring network forming a Distributed Hash Table (DHT) [32]. These participants engage in *knockout* tournaments to win and get elected as block proposers. A tournament in Colosseum consists of  $\log_2 N$  ( $N$  is the number of participants) asynchronous rounds. Nodes play matches in pair of two for each round and accept the winners of  $\alpha$  ( $\alpha < \log_2 N$ ) number of rounds as the proposers for that tournament. The elected proposers can then propose a block for the blockchain and propagate them in the network along with their *Proof-of-Win* (PoWin). PoWin is a cryptographically secured, tamper-proof certificate designed to get communicated and stored as the outcome of matches in Colosseum. The receivers validate the PoWin and the proposed block and push it into the ledger.

Members of Colosseum compete in a fair and randomized two-player game to select the winner and rely upon arbitrary members on the ring to validate and verify the result. Each match in the tournament consists of four stages. First, a node finds an eligible competitor for the match by sending TCP requests [20] to other members. Eligible players respond positively providing the PoWin of their previous round while others acknowledge negatively. Second, players need to create their *Game Proposal* for the two-player game. Cryptographic techniques are used to ensure that the

nodes generate random but verifiable game proposals. It helps to have a fair winner as none of the participants can falsify the process. Third, competitors send their proposals to a randomly selected *validator* on the DHT, which is unambiguously identifiable for a particular match. It validates the proposals of both the players and constructs the resulting certificate (i.e., PoWin) by comparing the proposals. Fourth, the result is sent back to the players and their *keepers* by the validator. Multiple keepers are assigned to both the participants of a match to store their state on the network. These keepers follow a weak consistency approach where the validator sends the results to some random keepers and the keepers then gossip it to other keeper nodes. Moreover, nodes on receiving the result also forward it to the keepers to maintain the consistency among all the keeper replicas.

Finally, winners move forward to play the next round or propose a block if they meet the qualifying condition (i.e., have won  $\alpha$  number of rounds), whereas the losers wait for the start of the next tournament. Simultaneously, keepers verify the results received by them with the keepers of the previous round. It is required to make sure that only the winners of a round proceed to the next round and vote against the players playing unauthorized matches. A match on receiving more than two-third of negative votes from its keepers is considered as a *foul*. These fouls impact the priority of the blocks proposed by nodes and lowers their chances of being in the final ledger.

### 4 Design

Converging Directed Acyclic Graph (CDAG) is proposed as an alternate data-structure to store data of a distributed ledger across a peer-to-peer network. It is similar to the traditional blockchain as it can be visualized as a chain and inspired from blockDAG to handle multiple simultaneous blocks. CDAG introduces Converging Block (C-Block) as a new construct to the structure of the ledger. C-Block is a converging point for the blocks in CDAG unlike other blockDAG protocols which do not have any single point of convergence (i.e., a block such that others are either before it or after it, making it a confirmation point for all previous blocks and last point of reference for the future blocks). Moreover, CDAG is introduced as an easy to understand and maintain structure, with more straightforward conflict resolution technique, which makes it desirable for blockchain systems.

Timeline in CDAG is divided into slots enforced by a *distributed time barrier* (DTB). Each tournament of Colosseum directly maps to one of the slots created by DTB. Multiple block proposers selected in one slot extend CDAG by proposing blocks from random buckets of transactions. They make the blocks consistent with the previous heaviest branch and disseminate in the network. The *barrier* between the two time slots provides adequate time for the proposed blocks

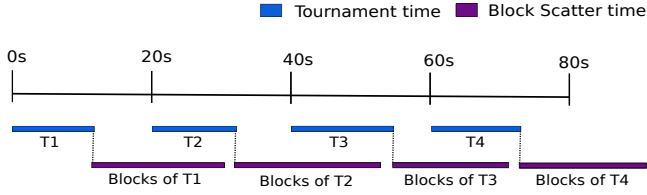


Figure 1. Timeline in CDAG with  $\tau = 20s$

to scatter in the network to increase their chances of converging at the next step; whereas bucketing of the transactions prevents them from generating intersecting blocks (blocks with the same transactions) because all of them have the same set of unconfirmed transactions.

#### 4.1 Distributed Time Barrier

As the name suggests, *Distributed Time Barrier* (DTB) is a time-based barrier to force the nodes of a network to halt for a particular time. Having a barrier allows CDAG to generate multiple simultaneous blocks since they will get the required time to scatter, and also restrict the nodes from moving ahead and generate more blocks until allowed. It plays a significant role in the convergence of CDAG at each stage, because, better the blocks propagate in the network more is their chance to get added to the ledger.

DTB maps each time slot exclusively to a tournament of Colosseum. Nodes need to wait for  $\tau$  seconds after the start of a tournament to play the next tournament. Figure 1 shows the division of time into slots of 20 seconds and each slot initiating a new tournament. Nodes participate in these tournaments to qualify and propose blocks for the ledger. The proposed blocks can propagate until the proposers from the next tournament get selected, as shown in figure 1. It gives them approximately  $\tau$  amount of time to scatter in the network irrespective of the time of a tournament but is valid only if the average completion time for the tournaments is less than  $\tau$  and multiple consecutive tournaments do not overlap.

CDAG uses Proof of Elapsed Time (PoET) [15] to enforce waiting on the nodes. PoET with the help of Intel Software Guard Extension [16] provides a framework to run trusted code on an untrusted host, which others can verify. Nodes need to provide the PoET certificate of waiting  $\tau$  seconds after the start of tournament  $t$  to play tournament  $t + 1$ . Each certificate contains a hash used to initialize the waiting for the next slot. These hashes are linked to each other such that nodes can start waiting for a particular tournament only after the successful generation of the PoET certificate for the previous tournament. Competitors verify these certificates while pairing to decide whether to play a match or not. It restricts the participants from moving ahead of others and thus fulfill the requirement of a time barrier across the network. The barrier makes them advance synchronously

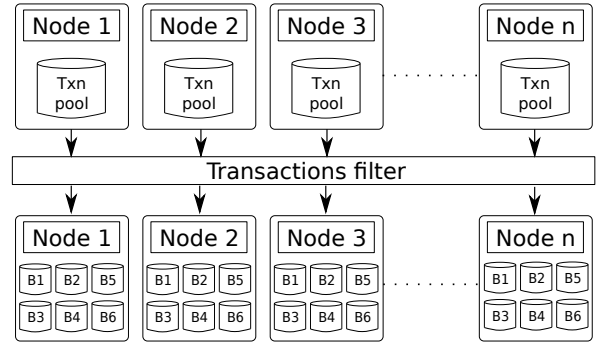


Figure 2. Bucketing of transactions

tournament-by-tournament and grants equal chance for the blocks of all the proposers of a tournament to reach a majority of the users.

However, nodes in CDAG need not be completely synchronous. Time difference of a few milliseconds (500 - 1000 ms) between the participants does not affect the tournaments since  $\tau$  is set sufficiently high. Still, as the process of waiting at the barrier is independent, clocks of players can drift apart further. Such nodes lacking behind need to poll the trusted oracles present in the network to reinitialize the state of their barrier. These trusted oracles also wait at the barrier along with the network and provide the bootstrap information for the trusted code precisely at the time of the start of the next slot. It also works similarly for the newly joined nodes in the system to get the bootstrap information and synchronize with others.

#### 4.2 Bucketing of transactions

CDAG generates multiple blocks in each slot concurrently, and there is a high probability of the same transactions being included by multiple proposers in their blocks because of a single pool of unconfirmed transactions. Proposers can unknowingly produce conflicting blocks even in the absence of double-spending transactions and can increase the orphan rate (the percentage of proposed blocks not included in the final chain) of the protocol. [25, 27] encounters the same issue because of their high block generation rate.

Transactions in CDAG are distributed among different buckets to counter this problem. The unconfirmed transactions are split across  $B$  non-intersecting transaction pools by the nodes. Each transaction maps to a single bucket based on its hash, as shown in equation (1). Proposers randomly select a bucket and include transactions only belonging to the particular bucket to generate a valid block. Blocks that belong to different buckets are assured of having different transactions and thus, limits the probability of the generation of conflicting blocks. The number of buckets is kept sufficiently high from the expected number of proposers to avoid any collisions in bucket selection. Also, proposers propagate the block headers in the network before sending

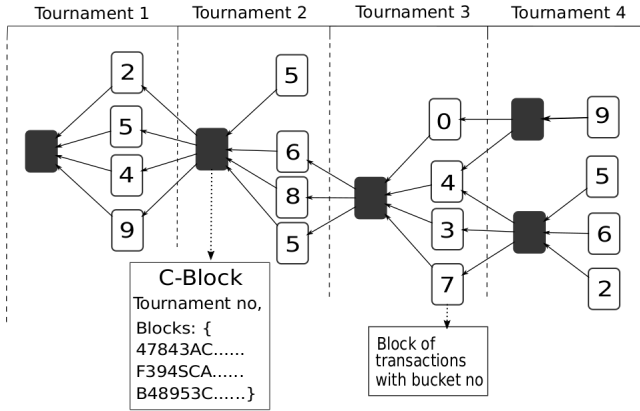


Figure 3. Structure of CDAG

the actual block to inform other proposers about the choice of the bucket. Block headers are comparatively smaller than the actual block and spread faster across the network, further minimizing the chances of creating conflicting blocks.

$$\text{Bucket Id} = (\text{Transaction Hash}) \bmod B \quad (1)$$

However, dividing transactions merely by their hashes do not protect the system against double-spending conflicts. Two transactions trying to spend the same money can have different hashes and may end up in different buckets. Therefore, blocks from different buckets can also conflict. CDAG handle such conflicts while converging these blocks and makes sure that it does not add multiple blocks containing double spending transactions to the ledger. Thus, the protocol works on its full potential as long as the users do not double-spend and can resolve the conflicts efficiently in comparison to others by using the heaviest branch selection mechanism of CDAG.

## 5 Converging DAG

CDAG is a blockDAG based distributed ledger which converges at each step to provide finality to the blocks. It is a serialized DAG designed to support smart contracts and quickly arrive at conclusions in the presence of conflicting transactions. It tries to maximize the throughput of blockchain systems and better utilize network resources.

CDAG uses Colosseum to elect multiple block proposers for each time slot. These proposers select a random bucket to generate blocks consistent with a *Converging Block* (C-Block) and disseminate it into the network. CDAG constructs C-Block as a collection of the blocks proposed in a round and works as a single point of reference for the newly generated ones. It enables CDAG to progress as a chain and shares the same trust model as the traditional blockchain (blocks of transactions linked to each other progressing as a modifications resistant ledger). Progress as a chain can create

temporary forks, and thus, CDAG uses the heaviest chain selection approach to have faster conclusions.

### 5.1 C-Block

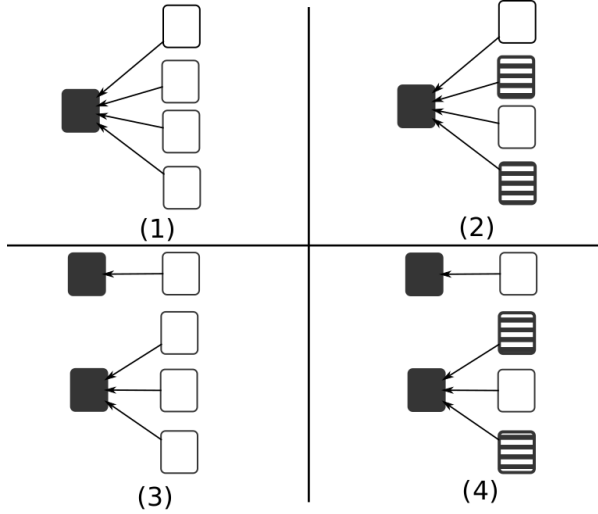
*C-Block* (Converging block) is the converging point in CDAG. It is introduced as an extra layer between two layers of blocks providing a point of convergence to the blocks of the previous slot and a single point of reference for all the blocks of the current slot, shown in Figure 3. As multiple blocks are recommended for the ledger in each slot, a C-Block refers to various non-conflicting blocks and contains the list of hashes of all the previous blocks it refers. A block consistent with a C-Block is bound to be consistent with all the blocks included in it. It allows the current block proposers to refer multiple blocks with a single link and enables different proposers to have the same view about the ledger.

Creation of C-Block is also the responsibility of the block proposers. While proposing a block, a proposer needs to make it consistent with a C-Block and includes a pointer to it. To do so, a proposer can either generate a new C-Block or use an existing C-Block created and propagated by other selected proposers of the same tournament. In case of a conflict, the heaviest chain rule is used to select the C-Block with higher probability to survive.

Generating a valid C-Block is a trivial process, and obeys a specific set of rules. Proposers need to make sure that all the blocks included in a C-Block must: 1) belong to different buckets, 2) have non-conflicting transactions, 3) refer to the same previous C-Block. The first rule ensures that a C-Block does not contain conflicting blocks due to the presence of the same transactions. Second does not allow double spending transactions in a C-Block, and third provides the serial nature to CDAG. Blocks of a tournament with different previous C-Blocks are similar to a fork in the traditional blockchain. In figure 3 the block of bucket 9 points to a different C-Block than that of buckets 5, 6, 2 at tournament 4, i.e., the previous blocks of both the C-Blocks are different, creating a fork. Thus, the structure progresses as a chain where a C-Block can be seen as a large block encapsulating all the blocks included in its list.

### 5.2 Total ordering for Smart Contracts

CDAG is designed as an alternate to other blockDAG based blockchain protocols to incorporate smart contracts. Phantom [27] also supports smart contracts by providing ordering between the blocks but is not as intuitive as provided by CDAG. Blocks in CDAG are ordered at two levels, and together it ensures a total ordering among all the blocks added in the main branch of the ledger, similar to a bitcoin blockchain. First, the arrangement of C-Block provides a higher level order in CDAG. Since CDAG extends as a chain of C-Blocks, blocks included in the predecessor (C-Block) of a block are inherently behind the current block. Second,



**Figure 4.** Possible leaf blocks formation while the generation of C-Block where Blocks with horizontal stripes conflict with each other.

a lower level ordering is provided in between the blocks included in a C-Block. These blocks are ordered based on their block hash resulting in an inter-block ordering. Such blocks do not refer to each other but are non-conflicting by nature as they are present in the same C-Block. Together both the properties provide the required total ordering at block level among the blocks of CDAG.

### 5.3 Heaviest chain selection

Heaviest chain selection in CDAG follows a greedy approach towards extending the chain with the highest probability to be the main chain. [26] motivates the chain extension algorithm used by CDAG, with a difference of trying to extend the chain of C-Blocks instead of regular blocks. The base assumption for the algorithm is that a majority of nodes in the system are honest and try to identify and report any unauthorized activity. It relies on the negative voting done against malicious nodes in Colosseum to identify honest blocks and branches. Votes against nodes reduce the priority of their blocks and therefore reduce its chance to be included in the main chain.

Each block in CDAG has an associated weight  $w$  which depends on the number of successfully verified matches of its proposer. Maximum weight for a block is  $\alpha$  corresponding to the  $\alpha$  number of wins gained to propose it. *Foul* (a match that is known to be unauthorized because of majority voting against it by the keepers of Colosseum) by nodes negatively affect the weight of these blocks. Each foul decrease the weight of the corresponding block by one unit resulting in low priority. Participants consider these votes while the generation of C-Block and decide whether to include a block or not.

The total weight of a chain  $W$  is the sum of the weights of all the blocks present in it. Nodes use it at two decision-making points during the tournaments. First, when a node needs to propose a block and has multiple C-Blocks to consider as a reference. It is similar to a fork in the blockchain, and the proposer needs to select the chain with the highest likelihood to survive. Second, when proposers need to generate a C-Block. Proposers need to maximize the weight of the chain they are proposing their block to by including the heaviest subset of non-conflicting blocks from the set of proposed blocks. In such a situation there can be multiple cases as shown in figure 4, 1) A single chain with no conflicting blocks; 2) A single chain with conflicts; 3) A forked chain with none of the chain having conflicting proposals; 4) A forked chain with conflicting block in both the chains or one of the chain. The first case is simple, where the proposer includes all the blocks while constructing the C-Block. In the second case, the proposer needs to select the heavier block among the conflicting blocks to include to the chain. In the third case, the proposer selects the heaviest of the multiple forks to construct the ledger, whereas in the fourth case the proposer first sorts the conflicts in-between the forks and then select the heaviest branch amongst them. Moreover, in the case when multiple conflicting blocks have the same weight, the block of the proposer with a smaller Id is considered. It ensures uniform generation of C-Block across the network, given all of them have received the proposed blocks and the negative votes by the keepers.

## 6 Block Proposal

Nodes with  $\alpha$  number of wins in a tournament of Colosseum are eligible to propose a block for CDAG. First, they select the heaviest C-Block to mark it as a reference in the new block, then proposes a block referencing the selected C-Block as the previous block. It attaches its latest PoWin (of the round number  $\alpha$ ) in the block header and propagates it into the network.

Each tournament of Colosseum can have a maximum of  $\log(N)$  ( $N$  is the number of nodes) rounds; therefore,  $\alpha$  is always set as  $\alpha < \log(N)$  to have multiple block proposers for a tournament. CDAG tries to commit all the non-conflicting blocks simultaneously using CDAG. It helps to utilize the bandwidth of the network better and decrease resource wastage. Blocks proposed by different nodes from distinct parts of the network propagate simultaneously, as compare to a single block getting propagated. It allows them to scatter in the entire network in almost the same amount of time as a single block without any extra cost. The distributed time barrier across tournaments also supports the design decision as it provides adequate time for the blocks to scatter in the network.

## 6.1 Multiple blocks per tournament

Allowing multiple block proposers to propose concurrently can escalate the generation of conflicting blocks because of the same unconfirmed transaction pool. Blocks can conflict in two ways. First, when two different blocks have an intersection (i.e., contains the same transactions). Second, when there is a double-spend across two blocks. The second one is a deliberate attempt, but the first issue occurs only because of the delays in the network. A proposer not aware of the blocks proposed by others can unknowingly generate a conflicting block.

CDAG uses bucketing of transactions to minimize the probability of the generation of intersecting blocks. However, bucketing does not resolve the conflicts that occur due to double-spend transactions. Such conflicts are resolved during the generation of C-Block by the block proposers of the next tournament. A C-Block contains the pointers to the previous blocks and is the point of reference for the newer blocks (similar to a previous block in blockchain). A valid C-Block does not contain pointers to multiple conflicting blocks. Thus, as long as a majority of the nodes extends the heaviest chain with valid C-Blocks, a double-spend cannot be successful.

Generating multiple blocks simultaneously also elevate the problem of having total ordering between the blocks. Blocks introduced concurrently or at small interval are unaware of the other proposals and do not align in an order. The design of C-Block handles this problem and also provides order to the structure. Since a C-Block collects only non-conflicting blocks into it and their hashes are arranged in sorted order, multiple blocks included to CDAG in a single slot are inherently ordered. Moreover, a higher level order is provided by the arrangement of CDAG as a chain of C-Blocks resulting in a total order among the blocks.

## 6.2 Waiting time for Distributed barrier

Each participant of the network waits for  $\tau$  amount of time after the start of a tournament to play the next tournament. Choice of  $\tau$  severely impacts the performance of the protocol since multiple blocks are being generated simultaneously. A small value of  $\tau$  does not provide enough time for the proposed blocks to reach everyone in the network. It can lead to a fork in the chain as there exist multiple combinations in which a node can commit blocks to the ledger. Proposers of the next slot will propose blocks according to their view of the network, which can lead to an increase in the orphan rate of the protocol and also affect the latency. Hence, CDAG prefers to have a large enough value of  $\tau$  because with multiple blocks per tournament and not enough time to propagate forking can be harmful to the safety of the protocol. It is also a significant reason to impose waiting on the nodes explicitly with the help of distributed time barrier.

The value of the waiting time ( $\tau$ ) depends on external as well as internal factors. External factors are the parameters that do not depend on the protocol like bandwidth, the number of nodes and internal factors are the tunable parameters of the protocol, such as  $\alpha$  and block size.  $\tau$  is directly proportional to the number of nodes and block size. The number of nodes in the network and the size of the block both increase the time required for the blocks to scatter in the network. Larger network size increases the number of matches played in the tournament to qualify which also requires more time.  $\tau$  is inversely proportional to the network bandwidth as higher bandwidth decreases the tournament completion and block scatter time. However, the relation between  $\tau$  and  $\alpha$  is not straightforward. The time required for higher values of  $\alpha$  increases because of the increase in the number of matches per tournament, and decrease because the number of qualifying nodes per tournament will drop, resulting in fewer blocks. Indirectly it depends on the external factors (i.e., bandwidth and number of nodes) to decide whether the overall effect is positive or negative.

## 7 Security Analysis

Securing blockchain systems is a significant concern. Malicious nodes can try to undermine the security of the system for their gain. Therefore, a blockchain protocol must be able to tackle adversaries and continue to work as long as the majority of the nodes in the system are honest. This section reviews such problems and discusses the countermeasures adopted in CDAG and Colosseum.

### 7.1 Irreversible transaction confirmation

Transaction confirmation is an essential aspect for all the blockchain systems. A transaction, once confirmed, must be irreversible to avoid a double spend. In bitcoin blockchain, confirmation of a block depends on its depth in the chain, but CDAG has a subtle difference. Multiple blocks get added to CDAG in each time slot, and the deepest chain may not be the one to which the majority of the participants are contributing. Therefore, Colosseum introduces the concept of “*full-confirmation*” for CDAG. A block is said to receive one full-confirmation when it has  $\delta$  (maximum expected blocks or block proposers from a tournament) number of blocks ahead of it. Equation (2) shows how to compute  $\delta$  for a given network where  $N$  is the number of nodes and  $\alpha$  is the number of wins required in Colosseum to propose a block.

$$\delta = \left\lceil \frac{N}{2^\alpha} \right\rceil \quad (2)$$

A block can get a full-confirmation in a minimum of one and a maximum of  $\delta$  number of tournaments. It is because the maximum number of blocks that can be added to the ledger in a slot is  $\delta$  and the minimum is 1. The blocks proposed in a tournament can have conflicts or may not reach

everyone in the given time slot due to network delays resulting in a variable number of blocks getting added to the ledger. In such cases, a full-confirmation may span up to multiple tournaments.

In CDAG, a block is considered as confirmed when it receives  $x$  full-confirmations in less than  $2x$  number of tournaments. It ensures that on an average more than 50% of the expected blocks (one half of  $\delta$ ) are added to the branch in which the block is present for each tournament (i.e., a majority of the network is contributing to the current branch continuously for a set of previous tournaments) and a hidden chain heavier than the current chain does not exist. It also makes sure that a fork cannot reverse a confirmed transaction because it needs more than  $\delta/2$  number of blocks per tournament for almost  $2x$  number of tournaments. In both cases, greater than 50% of the nodes (i.e., majority of the network) are required to extend the malicious chain. Therefore, as long as the majority of users in Colosseum are honest, a valid transaction cannot be reversed.

## 7.2 Byzantine faults

Nodes in the system can go rogue. They can perform malicious activities which favors them to take control of the system. CDAG depends on Colosseum for the selection of block proposers and can reasonably affect the progress and security of CDAG. Users play three roles in Colosseum: Player, Validator and Keeper. As a player, a malicious node can try to cheat and win, whereas validators and keepers can avoid their tasks of validation and verification to make other nodes suffer.

**Malicious Player.** *Players* in Colosseum need to find other players and contest in a match. These players can try to play multiple matches for the same round with different opponents in order to increase their chances of moving to the next round. Keepers in Colosseum can easily detect such matches and can alert other nodes in the system with verifiable proof. Result of the matches of the same round will land up with the same keeper replicas nodes and results in verifiable proof propagated in the network.

**Malicious Validator.** A *validator* in Colosseum collects the proposals from the players, compute the result and acknowledge with a Proof-of-Win. He is also responsible for sending the result to the *keeper*. In the case when an adversarial node is selected as the validator he may choose, 1) Not to reply, 2) Send result to players but not the keepers, 3) Send result to the keepers but not the players, 4) Send result to only one of the players and not to keepers, 5) Send delayed reply to the players.

In the first case both the nodes will timeout, check with the keepers (both their and opponent's) if they got the result and move forward to play the same round with new

competitors on being unsuccessful. The second case can result in keepers voting against the players for not receiving the proof-of-win. To prevent it nodes on receiving the result forward it to some of their keepers and the keepers gossip it forward. The third case is indistinguishable from the first case for the players as they will timeout and poll the keepers for the result, but this time the query will be successful. Fourth further gets divided into two subcases based on the player receiving the result is a winner or a loser. If the player is a winner, he will forward the certificate to its keepers such that they do not vote against him. The second player will find this Proof-of-Win (because same result certificate for both the players) from the keepers and will not proceed forward to play the same round again. On the other hand, if the receiving node was a loser and does not share the result with the keeper, both the nodes can play the same round again as nobody in the system knows about the match other than the losing node and the malicious validator. It is not fair to both the players but does affect the safety of the protocol. Also, it is similar to the first case when none of the players receives the result and both of them proceed. Also, random validator selection makes sure that such scenarios do not occur frequently.

The fifth case is more hazardous than others for honest players in Colosseum. A player waiting for a response from the validator may timeout and start finding a new opponent for the same round again. This situation is indistinguishable from the case when a player is trying to play multiple matches for the same round and can lead to the detection of an honest node as malicious. To prevent this situation of having a false-positive an honest node proceeds if he wins both the matches, i.e., he is the winner of the match whose result is received after the timeout and also wins the second match played. In case he lost the first match, he was intended to stop playing, which is fair and happens the same in this case. However, if he wins the first match and loses the second, he still needs to stop moving forward to not to get marked as malicious by the participants. Whereas, a node with both wins can play further and is not considered malicious. It is because, if a player knows that he has won, he will not try to play for the same round again as doing this can mark it as malicious if he loses by any chance. Therefore, keepers consider it as an honest mistake and do not send alerts in the network. However, it does not protect the malicious players trying to play multiple matches for a round as they need to win all of them to proceed in the tournament.

**Malicious Keeper.** *Keeper* plays the role of storing the state of nodes on the network, verifying their previous state and notify the network of any unauthorized activity. Multiple random nodes are selected as the keepers of a match to have a fault tolerant storage. However, one or more of these keepers may get compromised and do not perform their tasks



correctly. A keeper may, 1) Not verify a match whose PoWin is received, 2) not store the state with it, 3) send false alerts about unauthorized plays, 4) vote against matches of honest nodes. Most of these problems for the keepers of Colosseum gets solved because of their replication across the network. The honest counterparts will verify matches not verified by a malicious keeper. The state is stored at multiple nodes distributed over the entire network, and therefore, votes of a few dishonest nodes do not affect the final result for the players. Also, the keepers cannot generate false alerts as an alert requires proof of unauthorized play (multiple Proof-of-Win for the same round or Proof-of-Win for a higher round number of a player already lost in previous rounds).

Moreover, malicious nodes can masquerade as keepers and validator nodes for the matches because all the nodes in Colosseum are arranged in a DHT and are generally responsible for storing data for a range of keys. They can try to generate PoWin’s and cast votes on behalf of others. Colosseum uses a simpler form of DHT where nodes know all the other nodes present on the ring, making it challenging to masquerade others. Also, an honest node can easily detect and alert the system against such messages as the expected churn rate for the network is less and the participants are mostly known. Furthermore, researchers have extensively studied these topics in the past, and there are various solutions like [5, 30, 31], any one or a combination of them can be used with our protocol to improve and secure the messages over the DHT.

### 7.3 Bypassing Distributed time Barrier

Recent works (e.g., [3, 4]) suggest that bypassing the trusted execution environments (TEE) and undermining the verification process of PoET is possible. Colosseum assures that as long as a majority of the nodes in the network are honest, the protocol continues to work synchronously.

A match between two players depends on random validators and keepers. These nodes do not process a message if the difference between the tournament number of the user’s current state and any received message is more than one. Therefore, it is difficult for malicious players to enter a tournament not started for others. Dependency on other nodes indirectly forces everyone to move collectively. Thus, the model supports the idea of the distributed barrier and prevents adversarial nodes from undermining its entirety.

### 7.4 Liveliness vs Safety

A consensus protocol cannot guarantee both liveliness and safety for asynchronous networks, as stated by the FLP impossibility result [12]. CDAG chooses liveliness over safety. It guarantees liveliness as long as the majority of the selected proposers for the tournaments of Colosseum are honest and extend the heaviest chain of C-Blocks. A transaction in CDAG gets committed if the majority of the network is

Parameter	Meaning	Value
$\alpha$	number of wins required to propose a block	4
$K$	keeper replication factor	16
$\tau$	Distributed barrier delay in sec	20 sec
$B$	number of buckets of transactions	40
$F$	minimum full confirmations to confirm a block	3

**Table 1.** Tunable parameters in the protocol

working towards extending the heaviest chain, as described in section 7.1, and more than one branch cannot have majority of blocks simultaneously makes sure that eventually blocks get confirmed and nodes reach to a consensus, but not in the case of network partitions. However, safety is not guaranteed by CDAG because temporary forks can occur, i.e., two honest nodes can have different truths at some point in time but get resolved irreversibly.

## 8 Implementation

We implemented CDAG and Colosseum in JAVA consisting of all the elements that are significant to analyze its performance. For the underlying network, we use Vishwa [21] a peer-to-peer grid computing middleware. It uses ZonalServer (trusted oracle in case of Colosseum) to manage the underlying structured network in Colosseum which forms a DHT and provides bootstrap information to newly joined nodes. Colosseum uses a similar network model and is, therefore, a perfect match.

In our implementation user on receiving a message gossips it to three nodes (two random nodes from the routing table and an immediate successor) in the structured layer. All the messages are digitally signed and also contain the public-key certificates of the user (we can also broadcast public-key certificates initially to save the overhead). We use waiting to simulate the behavior of PoET for the distributed time barrier. A block contains the header, PoWin and dummy data instead of actual transactions.

Table 1 shows all the tunable parameters in the protocol with their default values unless specifically mentioned for experiments. We set the minimum number of full-confirmations required to consider a block as confirmed to 3. The keeper replication factor is set to 16 for all the experiments but depends on the scale of the system.

## 9 Evaluation

We evaluate the performance of CDAG for different sizes of network and vary the percentage of malicious users present in it. The experiments measure the throughput and show the variation in the latency and orphan rate of the protocol for three different configurations, as shown in table 2. The results demonstrate that how CDAG scales with the number of nodes by tuning some hyper-parameters and still give

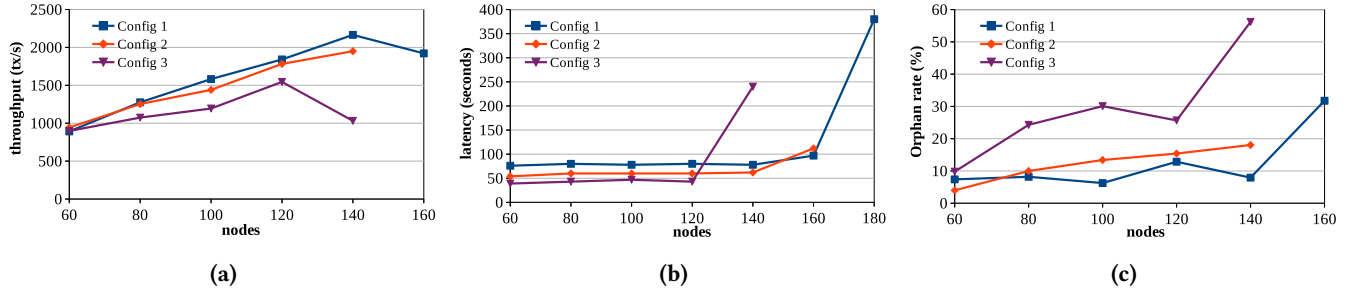


Figure 5. Throughput, Latency and Orphan rate of CDAG for  $\alpha = 3$

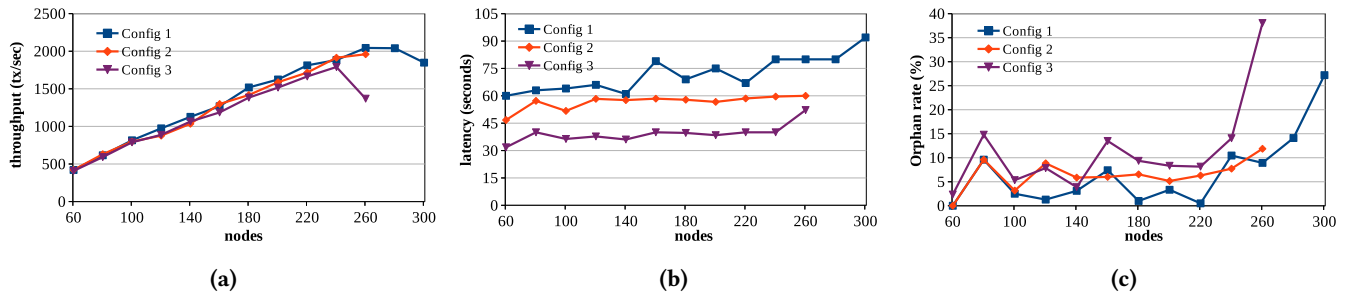


Figure 6. Throughput, Latency and Orphan rate of CDAG for  $\alpha = 4$

stable results. We also measure the performance of CDAG under the influence of adversaries in the network and show its effect on some of the performance parameters.

Configuration	Block Size	Time slot ( $\tau$ )
Config 1	1 MB	20s
Config 2	0.75 MB	15s
Config 3	0.50 MB	10s

Table 2. Different configurations used in experiments

We deploy CDAG on Google Cloud Platform [7] using Google Kubernetes Engine [8]. The Kubernetes cluster consists of multiple n1-highmem-16 instances, each with 16vCPUs and 104 GB memory. Participants of CDAG are encapsulated in docker containers [18] and deployed on the cluster with an upper and lower limit of twelve and eight docker containers per instance and bandwidth constraint of 25 Mbps per docker container. We also make sure that multiple dockers on the same instance also adhere to the bandwidth constraints. Users propose blocks of different sizes for different set of configurations with 1 MByte equivalent to 2500-3000 transactions. Each experiment runs for at least 30 time slots and the graphs in this section plots an average over all of them.

### 9.1 Throughput

To measure the throughput (transactions/sec) of the protocol, we vary the number of nodes in the network for multiple values of  $\alpha$  and all the three configurations as mentioned in table 2. Both the graphs 5a and 6a shows that for a fixed value of  $\alpha$  the throughput of the system increases initially with the number of nodes and then starts decreasing. The initial increase is because the number of blocks for a tournament of Colosseum  $\delta$  is directly proportional to the number of nodes, as shown in equation (2). More number of nodes results in a higher degree of parallelism for CDAG and thus, increases the throughput. After a certain point, the throughput starts decreasing for all the different configurations in 5a, 6a due to the generation of a large number of simultaneous blocks. Bandwidth constraints and network congestion do not provide them enough time to reach everyone in the network. Also, it is evident in both the graphs that the configuration with smaller time slot  $\tau$  (e.g., Config 3) saturates before than the ones with a higher time slot (e.g., Config 1) even though the size of the block also increments in the same ratio.

Across 5a and 6a the throughput of CDAG decrease for the same network configuration. It is because, with the increase in the value of  $\alpha$ , the number of qualifying proposers  $\delta$  for a time slot decreases, as inferred from equation (2). However, the saturation point shifts to a higher number of nodes for larger values of  $\alpha$ . It allows CDAG to scale with the number of nodes. The value of  $\alpha$  can be configured to achieve the maximum throughput for a given configuration

of the network and can change with time similar to setting the hardness of the cryptographic puzzle in Bitcoin.

Results show that CDAG achieves a throughput of approx 2100 transactions/sec with a network size of approx 300 nodes and can also scale further. It is 28.57% greater than the on-chain transaction throughput of 1500 transactions/sec [22] of ripple with the validator set ranging from 30 to 50 peers [23]. On the other hand, Hyperledger [1] shows a throughput of 3500 transactions/second for about 100 peers. However, it uses very highly configured peers with 16vCPUs and 1Gbps network link in the experimental setup, unlike CDAG which is tested using very nominal resources as discussed earlier in this section.

## 9.2 Latency

Figure 5b, 6b shows the block confirmation time for different configurations (ref table 2) with the increase in the number of nodes in the network. It is evident that the block confirmation time for configurations with smaller  $\tau$  is generally lower in comparison to those having a large value of  $\tau$ . It is because the minimum confirmation time for a block is  $F * \tau$ , where F is the minimum *full-confirmations* required to confirm a block and  $\tau$  is the minimum time to get one full-confirmation.

Network saturation also affects the latency of the protocol, similar to what was observed for throughput. Increasing the number of block proposers (results from increasing the number of nodes for a particular  $\alpha$ ) beyond a limit lowers the percentage of blocks getting added to the ledger for each slot, resulting in an increase in the number of slots required for a block to achieve one full-confirmation and therefore, increasing the latency. Between 5b and 6b, the network with a larger  $\alpha$  has much more stable latency values for larger network sizes as a result of the lower network traffic due to lesser blocks in each slot. The saturation point also moves forward for a higher value of  $\alpha$ . It indicates towards a scalable blockchain protocol with acceptable latency values in the order one minute.

## 9.3 Orphan rate

Orphan rate of the protocol shows a behavior similar to the latency of CDAG since both of them are correlated. Latency in CDAG is derived from the percentage of blocks getting added to the main chain with respect to the maximum, which defines orphan rate. In figure 5c and 6c the orphan rate is initially in a stable range with the increase in the number of nodes and then shoots up because of the network congestion similar to figure 5b and 6b.

The duration of the time slot  $\tau$  significantly affects the orphan rate of CDAG as providing sufficient time for the blocks to scatter in the network better utilize the network resources. Figure 5c and 6c shows that the orphan rate is largely lesser for the configuration with the higher values of  $\tau$  because of the higher stability achieved by the network

for each slot. A larger  $\tau$  allows more number of concurrent blocks to reach everyone in the network and get added to the heaviest chain, resulting in a drop in the orphan rate. Some inconsistencies in the initial part of graphs are because of a few blocks getting generated for each slot and the randomness involved in the bucket selection process while the generation of blocks. Selection of similar buckets by the proposers in such cases can have a significant effect on the percentage of discarded blocks in comparison to stages when the block frequency is high.

## 9.4 Misbehaving Users

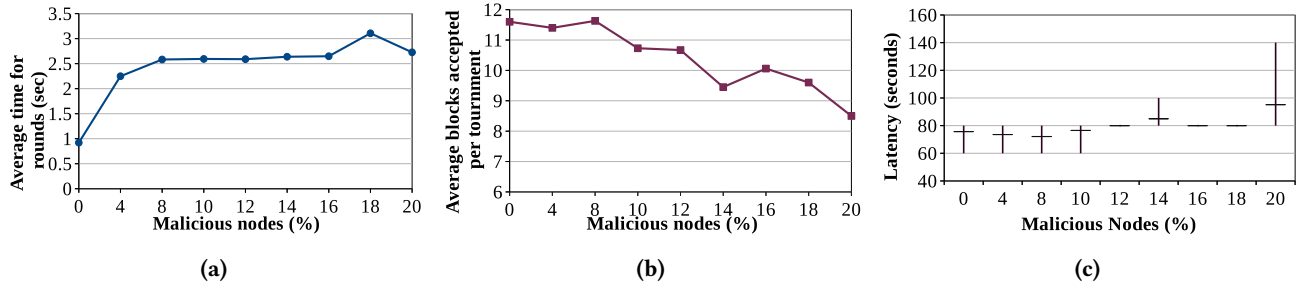
We show that Colosseum can handle malicious users by randomly adding some percent of them in the network. These users defy the protocol by not performing any of the tasks of validators and keepers, but themselves play to propose blocks. Figure 7a, 7b show the variation in the average time for the rounds of Colosseum and the number of blocks getting added to CDAG with the increase in the number of malicious players. The average time for individual rounds of a tournament increases with the number of malicious users and the average number of blocks accepted per tournament decrease. It is because the number of matches dropped by malicious validators increase and more blocks get discarded due to rise in the false-negative votes against matches of honest players by malicious keepers. A little inconsistency in results is because of the randomness involved in placing the nodes over DHT and selecting validators and keepers for the matches.

Figure 7c shows the minimum, average and maximum latency of Colosseum and its variation with the number of malicious users. It is evident that Colosseum confirms transactions in less than two minutes (120 seconds) when approx. 20% of validators and keepers are not performing their tasks honestly.

## 10 Limitations and Future Work

This paper discussed a new structure to store transactions over a distributed ledger similar to the blockchain. The structure suffers from temporary forks even if users propose non-conflicting blocks. It can be due to smaller time slots for the barrier or low network bandwidth. A blockDAG kind of approach on top of CDAG to converge multiple non-conflicting forked chains can help to decrease the orphan rate further.

We do not discuss optimal configuration setting for CDAG to achieve maximum throughput and minimum latency. A detailed analysis of the configuration parameters and discussion on the trade-offs can be helpful to gain performance and better utilize the resources. One can also explore game theory techniques to have easily verifiable multi-player distributed games to filter out the nodes and select the block proposals more efficiently and independently.



**Figure 7.** For a varying fraction of malicious users, (a) Average time for one round of the tournament, (b) Average number of blocks in each C-Block, (c) Min, Avg and Max transaction latency

## 11 Conclusion

CDAG is a new distributed data-structure to store transactions as an alternate to blockchain and blockDAG. It scales well with the number of nodes while maintaining the consistency in its performance. It uses Colosseum to select multiple block proposers in each time slot and use different buckets of transactions to generate non-conflicting blocks. Unlike other DAG based protocols, CDAG provides immediate finality to the blocks of the ledger and has a totally ordered structured by design. It is designed to have lower orphan rates for real-time scenarios and support smart contracts. Experimental results for Colosseum demonstrate that it achieves a throughput of more than 2000 tps and confirm transactions in less than two minutes.

## References

- [1] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. 2018. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. *arXiv preprint arXiv:1801.10228* (2018).
- [2] Belcher. 2018. Confirmation-Bitcoin Wiki. Retrieved 2018-11-05 from <https://en.bitcoin.it/wiki/Confirmation>
- [3] Andrea Biondo, Mauro Conti, Lucas Davi, Tommaso Frassetto, and Ahmad-Reza Sadeghi. 2018. The guard's dilemma: Efficient code-reuse attacks against intel sgx. In *Proceedings of the 27th USENIX Conference on Security Symposium*. USENIX Association, 1213–1227.
- [4] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. 2017. Software grand exposure: SGX cache attacks are practical. *arXiv preprint arXiv:1702.07521* (2017), 33.
- [5] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S Wallach. 2002. Secure routing for structured peer-to-peer overlay networks. *ACM SIGOPS Operating Systems Review* 36, SI (2002), 299–314.
- [6] Miguel Castro, Barbara Liskov, et al. 1999. Practical Byzantine fault tolerance. In *OSDI*, Vol. 99, 173–186.
- [7] Google Cloud. 2018. Google Cloud Platform. Retrieved 2019-5-16 from <https://cloud.google.com/docs/overview/>
- [8] Google Cloud. 2018. Kubernetes Engine. Retrieved 2019-2-21 from <https://cloud.google.com/kubernetes-engine/>
- [9] Christian Decker and Roger Wattenhofer. 2013. Information propagation in the bitcoin network. In *IEEE P2P 2013 Proceedings*. IEEE, 1–10.
- [10] Ethereum.org. 2018. Ethereum Project. Retrieved 2018-11-25 from <https://www.ethereum.org/>
- [11] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. 2016. Bitcoin-NG: A Scalable Blockchain Protocol. In *NSDI*, 45–59.
- [12] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. 1982. *Impossibility of distributed consensus with one faulty process*. Technical Report. MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTER SCIENCE.
- [13] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 51–68.
- [14] Himanshu Gupta and Dharanipragada Janakiram. 2019. Colosseum: A Scalable Permissioned Blockchain over Structured Network. In *Proceedings of the Third ACM Workshop on Blockchains, Cryptocurrencies and Contracts*. ACM, 23–25.
- [15] Hyperledger.org. 2018. PoET-Proof of Elapsed Time. Retrieved 2018-11-11 from <https://sawtooth.hyperledger.org/docs/core/releases/latest/architecture/poet.html>
- [16] Intel.com. 2018. Intel SGX. Retrieved 2019-1-28 from <https://software.intel.com/en-us/sgx>
- [17] Litecoin.com. 2018. Litecoin | Money for the Internet Age. Retrieved 2018-11-25 from <https://litecoin.com/>
- [18] Dirk Merkel. 2014. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal* 2014, 239 (2014), 2.
- [19] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. <https://bitcoin.org/bitcoin.pdf>
- [20] Jon Postel. 1981. *Transmission control protocol*. Technical Report.
- [21] M Venkateswara Reddy, A Vijay Srinivas, Tarun Gopinath, and D Janakiram. 2006. Vishwa: A reconfigurable P2P middleware for Grid Computations. In *Parallel Processing, 2006. ICPP 2006. International Conference on*. IEEE, 381–390.
- [22] Ripple.com. 2018. Ripple. Retrieved 2019-5-15 from <https://ripple.com/xrp/>
- [23] Ripple.com. 2018. Ripple. Retrieved 2019-5-16 from <https://xrcharts.ripple.com/#/validators>
- [24] Mooly (Shmuel) Sagiv. 2018. BlockDAG protocols SPECTRE, PHANTOM. Retrieved 2019-5-2 from [http://www.cs.tau.ac.il/~msagiv/courses/blockchain/PHANTOM\\_Sompolinsky.pdf](http://www.cs.tau.ac.il/~msagiv/courses/blockchain/PHANTOM_Sompolinsky.pdf)
- [25] Yonatan Sompolinsky, Yoav Lewenberg, and Aviv Zohar. 2016. SPECTRE: A Fast and Scalable Cryptocurrency Protocol. *IACR Cryptology ePrint Archive* 2016 (2016), 1159.
- [26] Yonatan Sompolinsky and Aviv Zohar. 2015. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security*. Springer, 507–527.
- [27] Yonatan Sompolinsky and Aviv Zohar. 2018. PHANTOM: A Scalable BlockDAG Protocol. *IACR Cryptology ePrint Archive* 2018 (2018), 104.

- [28] Nick Szabo. 1997. The idea of smart contracts. *Nick Szabo's Papers and Concise Tutorials 6* (1997).
- [29] Alexandra Tran. 2018. An Introduction to the BlockDAG Paradigm. Retrieved 2018-11-26 from <https://blog.daglabs.com/an-introduction-to-the-blockdag-paradigm-50027f44fad>
- [30] Peng Wang, Ivan Osipkov, N Hopper, and Yongdae Kim. 2006. Myrmic: Secure and robust dht routing. *U. of Minnesota, Tech. Rep* (2006).
- [31] Qiyang Wang and Nikita Borisov. 2012. Octopus: A secure and anonymous DHT lookup. In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*. IEEE, 325–334.
- [32] Wikipedia.org. 2018. Distributed Hash Table. Retrieved 2018-17-07 from [https://en.wikipedia.org/wiki/Distributed\\_hash\\_table](https://en.wikipedia.org/wiki/Distributed_hash_table)
- [33] Gavin Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper* 151 (2014), 1–32.
- [34] Z.cash. 2018. Privacy-protecting digital currency | Zcash. Retrieved 2018-11-25 from <https://z.cash>