2019 IEEE International Conference on Blockchain (Blockchain)

CoDAG: An efficient and compacted DAG-based blockchain protocol

Shu Yang[1], Ziteng Chen[1], Laizhong Cui[1*], Mingwei Xu[2], Zhongxing Ming[2], Ke Xu[2]
[1] College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, PR.China
[2] Department of Computer Science and Technology, Tsinghua University, Beijing, PR.China
Emails: yang.shu@szu.edu.cn, chenziteng1996@gmail.com, cuilz@szu.edu.cn
xmw@cernet.edu.cn, mzx@mail.tsinghua.edu.cn, xuke@tsinghua.edu.cn

***Abstract*—Blockchain is seen as a promising technology to provide reliable and secure services due to its decentralized characteristic. However, because of the limited throughput, current blockchain platforms can not meet the transaction demand in practical use. Though researchers proposed many new solutions, they suffered either decentralization or security issues. In this paper, using Directed Acyclic Graph (DAG) structure, we improve the linear structure of traditional blockchain protocol. In the new structure, blocks are organized in levels and width, which will generate into a compacted DAG structure (CoDAG). To make CoDAG more efficient and secure, we design algorithms and protocols to place the new-generated blocks appropriately. Compared with traditional blockchain protocols, CoDAG improves the security and transaction verification time, and enjoys the consistency and liveness properties of blockchain. Taking adversary parties into consideration, two possible attack strategies are presented in this paper, and we further prove that CoDAG is a secure and robust protocol to resist them. The experimental results show that CoDAG can achieve 394 transactions per second, which is 56 times of Bitcoin's throughput and 26 times of Ethereum's.**

***Index Terms*—Blockchain, Compacted Directed Acyclic Graph**

I. Introduction

Recently, blockchain has attracted attention from industry and academia. Participants in the network share the same ledger, making it a trustworthy and secure system without any centralized authority in distributed environment. Bitcoin [1] is the first blockchain protocol, where blocks are organized in a linear chain. To maintain the chain, miners in Bitcoin will try their best to solve random cryptographical puzzles, which is called *proof-of-work*, or *PoW* [2].

However, throughput of traditional blockchain protocols is relatively low [3], and researcher proposed different schemes to improve it. Among them, solutions (e.g., IOTA [4]) based on Directed Acyclic Graph (also DAG) allow multiple blocks to append the tail of the graph concurrently, thus they strike a balance between asynchronization and speed. However, as blocks append the network, security of IOTA will decrease, and transaction confirmation time will be non-deterministic. Although solutions including Phantom [5] and Spectre [6] try to improve performance and security, their data structures are complicated, and blockchain users prefer simple and deterministic structures.

This work is supported by the National Natural Science Foundation of China (Grant No. 61772345) and National Engineering Laboratory for Big Data System Computing Technology.

In this paper, we propose a **Co**mpacted **DAG**-based blockchain protocol CoDAG. Different from IOTA, we design a compacted DAG structure, where blocks in the network are organized in levels. Each level has fixed width which means the maximun number of blocks in the level. When a new-generated block joins the network, rather than placing it randomly, CoDAG will use algorithms to place it at an appropriate level. CoDAG improves the confirmation time and keeps a simple data structure at the same time. Meanwhile, we illustrate two attack strategies, and prove that CoDAG can resist them. Furthermore, we implement a CoDAG prototype, and simulation results show that CoDAG can achieve 394 transactions per second, which is 56× Bitcoin's throughput and 26× Ethererum's respectively.

II. Related Work

To achieve consensus, many ledger structures have appeared recently, which can be divided into permissioned protocol, permissionless protocol (also public blockchain and private blockchain) and hybrid protocol [7]. In permissioned protocol (e.g., PBFT [8]), only nodes with permission can join the network, and they will achieve fast consensus. While in permissionless protocol, nodes can freely join or leave the network. Nakamoto consensus [1] is the first permissionless protocol, but it has very limited throughput, e.g., 7 transactions per second.

To improve the throughput of permissionless protocol, many schemes are proposed, including **off-chain model** [9], [10] and **on-chain model** [4]–[6]. Among on-chain models, DAG-based solutions [4]–[6] are considered efficient approaches to address the throughput problem, which allow multiple blocks to join the network concurrently. IOTA [4] is famous for its fast, feeless and minerless characteristics, but it will cause non-deterministic confirmation time. Spectre [6] enjoys a high throughput and fast confirmation process, but it can not support full linear ordering over all blocks. Phantom [5] keeps a full ordering, but at the cost of non-deterministic confirmation process.

Moreover, hybrid consensus protocol combines the advantages of permissioned and permissionless protocols. However, they still rely on permissionless protocol as the backbone consensus.

978-1-7281-4693-5/19/$31.00 ©2019 IEEE
DOI 10.1109/Blockchain.2019.00049

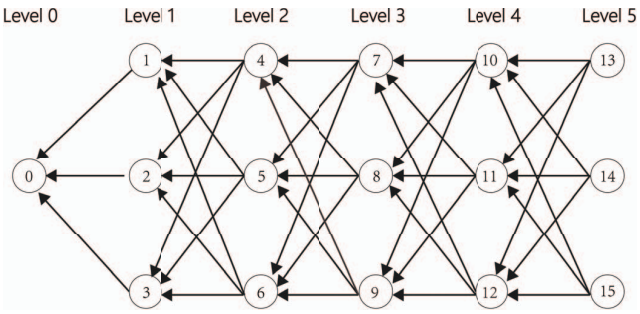


Fig. 1: A simple example for CoDAG

III. Basic Idea of CoDAG

In this document, we design a new ledger structure called CoDAG (**Co**mpacted **D**irected **A**cyclic **G**raph). Transactions are enveloped in blocks, which is organized as directed acyclic graph (DAG). The DAG is a gossip-like structure that fits well in asynchronous operations. Unlike previous DAG-based structures, the new graph is well connected between the honest blocks, and it has a limited width.

In the new ledger, let $G = (V, E)$ be a DAG, where V is the set of nodes or blocks (in the following text, we use node and block interchangeably), and E is the set of directed edges. In the graph, we use l_v to denote the level of a node in the graph, which is the length of the longest path from the genesis node to the current node. Each level has a width, which is the number of nodes in the level. For example, there are 3 nodes in level 3, and we use w_l to denote the width of level l. For a DAG, we call its width is constrained by a constant K, if $\forall l$, $w_l < K$, e.g., the DAG in Fig. 1 is constrained by 3. The width represents the maximum number of blocks the system can generate in a round. It could also be adjusted (or self-adaptive) according to the transaction generation rate in the network. In the worst case, the structure degenerates into Bitcoin's linear blockchain structure if the width equals 1.

In the graph, each node points to K nodes in the previous level, and we call the number of paths between a node v and another node w the connectivity between v and w, and use $C_{v,w}$ to denote it. The connectivity from a node to itself is set to be 1, i.e., $C_{v,v} = 1$. For each node v, we call the number of paths from genesis to it *connectivity* of the node, and use C_v for brevity. For example, in Fig.1, the connectivity of node 8 is 9.

We call the leaf nodes *tips*, and the tip that has the largest connectivity number is called the *navigator* of the graph. If there exist multiple tips with equal connectivity, then all of them are considered to be navigators. For example, node 13, 14, 15 have the same connectivity, and they are all selected as navigators. We call the number of paths from the navigators to the a node *reverse connectivity* of the node, and use R_v to express it. For example, the reverse connectivity of node 8 is 9.

IV. Protocol of CoDAG

A. Overview

Like Nakamoto consensus, each miner in CoDAG will generate blocks through solving puzzles, despite its much faster generation rate in CoDAG. Every new generated block is appended to CoDAG as a tip and validates multiple blocks in the previous level, i.e., points to multiple previous blocks in the graph.

Moreover, we set an upper limit K on the width of the graph. However, due to network delay or adversarial nodes, there may exist forks, i.e., more than K nodes in some level. In this case, nodes with higher reverse connectivity will have higher priority. The top K priority nodes in the same level are considered *candidates*. We use $\mathcal{C}_l$ to denote the candidate set of level l. The miner that generates a stable block will be rewarded by the system. A special case is that several nodes with the same priority belong to the borderline candidates. Under the circumstances, all these nodes will be regarded candidates and share the rewards.

B. Algorithms of CoDAG

Algorithm 1: Connectivity(G, v)

```
begin
    C_v ← 0, ∀v ∈ V;
    L ← maximum level of G;
    C_genesis ← 1;
    for l_v + 1 ≤ l ≤ L do
        for ∀v ∈ {v ∈ V | l_v = l} do
            for ∀w ∈ {l_w = l − 1} do
                if ⟨v, w⟩ ∈ E then
                    C_v += C_w;
    return {C_v | v ∈ V};
```

Algorithm 2: Generate($G = (V, E)$, x)

```
begin
    Connectivity(G, genesis);
    if |{v_L}| = K then
        L += 1;
    S ← K nodes that have the largest connectivity
      in level L − 1;
    V ← V ⋃ {x};
    E ← E ⋃ {⟨x, v⟩, v ∈ S};
```

Block Generation Algorithm: We use L to denote the maximum level of the graph G, and $\langle v, w\rangle$ to denote a directed edge that points from v to w. We design algorithm $Generate()$ (shown in Algorithm 2) to compute where a new generated block x is to be appended. Algorithm $Generate()$ first computes the connectivity number of each node, and points to those with largest connectivity in the previous level.

In $Generate()$, we use algorithm $Connectivity()$ (shown in Algorithm 1) as an intermediate function, which takes a DAG and a node as the input, and computes the connectivity from all nodes (behind it) to it.

Candidate Selection Algorithm: We design algorithm $Select()$ (shown in Algorithm 3) to select the candidate for some level in the DAG. $Select()$ computes the candidate node for a level in the DAG. It first computes the connectivity of each node, and the node with the largest connectivity is set to be the navigator. Secondly, for a specific level l, $Select()$ computes the reverse connectivity for each node in the level. At last, the nodes that have top K largest reverse connectivity are selected as the candidates.

Algorithm 3: Select($G = (V, E)$, l)

```
begin
    L ← maximum level of G;
    C_l ← ∅;
    Connectivity(G, genesis);
    N ← {v|C_v ≤ C_w, ∀w ∈ V} # there may be
      multiple nodes with equal connectivity;
    R_v ← 0, ∀v ∈ V, R_x = 1, ∀x ∈ N;
    for L − 1 ≥ k ≥ l do
        for ∀v ∈ {v ∈ V|l_v = k} do
            for ∀w ∈ {l_w = k + 1} do
                if ⟨w, v⟩ ∈ E then
                    R_v+ = R_w;
    T ← |{v|l_v = l}| ≤ K;
    if T ≤ K then
        return C_l ← T ;
    else
        sort(T, {C_v|v ∈ T}) #sort T from highest to
          lowest according to its reverse connectivity
          value ;
        for 0 ≤ i ≤ K − 1 do
            C_l ← C_l ⋃ {T[i]}
        return C_l ← C_l ⋃ {v|v ∈ T&R_v = R_{T[K−1]}};
```

V. Well Connected Channel of CoDAG

Different from chain-like structure, CoDAG adopts a flexible structure that is more suitable for asynchronous operations. The structure is compacted, and nodes are more well connected. Intuitively, CoDAG is more like a well connected channel, and the width of the channel can be adjusted according to user demands.

We consider the scenario where an adversary tries to generate an alternate channel with higher connectivity than the honest channel, and takes control of the navigator node. Like Bitcoin whitepaper [1], we define the following notations:

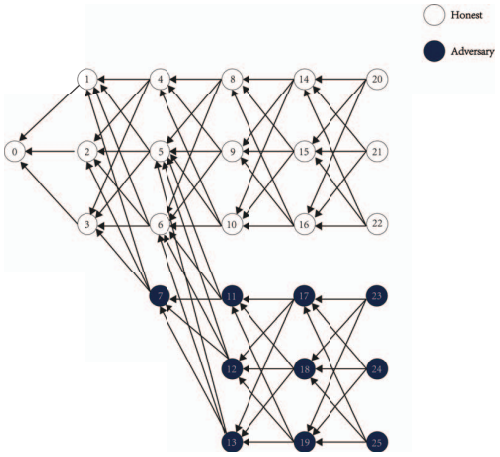
- p = the proportion of computing power that the honest party controls
- q = the proportion of computing power that the adversary party controls
- q_z = probability that the adversary will ever catch up from z levels behind

Lemma 1.

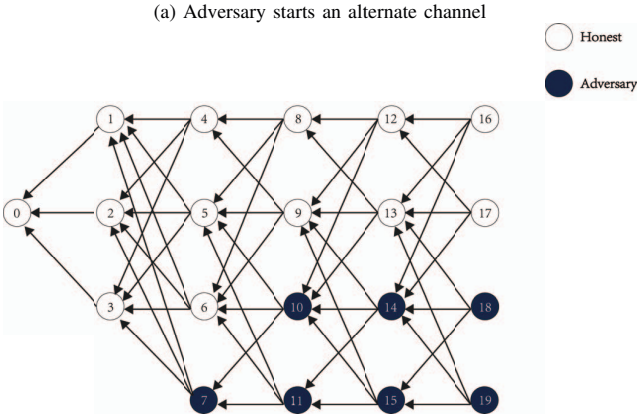
$$q_z = \begin{cases} 1, & \text{if } p \leq q. \\ (q/p)^z, & \text{otherwise}. \end{cases} \quad (1)$$

Basically, to control the navigator, the adversary needs to produce more nodes in the alternate channel than the channel produced by the honest miners. In each level, if both parties start simultaneously, the probability that the honest party first finds the next w block is p, and the probability that the adversary party first finds the next w block is q. Thus, the left proof is the same as [1].

Next, we consider how long a transaction will be confirmed. A node in some level selected as the candidate could be taken over by another node in the same level.



(a) Adversary starts an alternate channel



(b) Adversary competes in the same channel

Fig. 2: Two attacking strategies on CoDAG

Lemma 2. *If node v locates z levels away from the largest level, then the possibility that its candidate identity could be overtaken is:* $1 - \sum_{k=0}^{z} \frac{\lambda^k e^{-\lambda}}{k!}(1 - (q/p)^{z-k})$, *where* $\lambda = z\frac{q}{p}$.

Proof. In level l, we suppose there exist an honest node a and an adversarial node b, where $R_a > R_b$ and $L - l \leq z$.

Firstly, we will prove that $\forall k > l$, for two honest nodes v_i and v_j, then $C_{v_i,a} = C_{v_j,a}$ and $C_{v_i,b} = C_{v_j,b}$; similarly, for two adversary nodes w_i and w_j, $C_{w_i,a} = C_{w_j,a}$ and $C_{w_i,b} = C_{w_j,b}$. Suppose in level k, we have P honest nodes and Q adversary nodes, where the honest node v^k satisfies $C_{v^k,a} = X$ and $C_{v^k,b} = Y$, and the adversarial node w^k satisfies $C_{w^k,a} = X'$ and $C_{w^k,b} = Y'$. Then in level $k+1$, honest node v^{k+1} will satisfy $C_{v^{k+1},a} = X*P+X'*(K-P)$ and $C_{v^{k+1},b} = Y*P+Y'*(K-P)$, and adversary node w^{k+1} will satisfy $C_{w^{k+1},a} = X'*Q+X*(K-Q)$ and $C_{w^{k+1},b} = Y'*Q+Y*(K-Q)$.

Secondly, we will prove the lemma. For a level $k > l$, let $\mathcal{R}(k)$ be the total honest reverse connectivity, and let $\hat{\mathcal{R}}(k)$ be the total adversary reverse connectivity. Then in level $k+1$, $\mathcal{R}(k) = \mathcal{R}(k+1)*P+\hat{\mathcal{R}}(k+1)*(K-Q)$ and $\hat{\mathcal{R}}(k) = \hat{\mathcal{R}}(k+1)*Q+\mathcal{R}(k+1)*(K-P)$. Therefore, we get $\frac{\hat{\mathcal{R}}(k+1)}{\mathcal{R}(k+1)} \geq \frac{\epsilon*P+P-K}{Q+Q*\epsilon-\epsilon*K}$. Since P and Q are subject to independent poisson distributions, then $E(\frac{\hat{\mathcal{R}}(k+1)}{\mathcal{R}(k+1)}) \geq \frac{\epsilon*E(P)+E(P)-K}{E(Q)+E(Q)*\epsilon-\epsilon*K} \geq \epsilon*E(\frac{P}{Q}) = \epsilon*\frac{p}{q}$, where $E(\frac{P}{Q})$ is the ratio of honest computing power to adversary computing power.

□

There are two strategies in the channel-based scenario. The first strategy is that, the adversary will start an alternate channel, as shown in Fig. 2(a). The model then is the same as Bitcoin. The second strategy is that, the adversary competes in the same channel with the honest miners, as shown in Fig.2(b). The adversary will first point to adversary nodes, and honest nodes will first point to honest nodes. We note that if the honest party owns more computing power, most votes will be given to the honest nodes, and honest votes will get accumulated with the level growing.

VI. Implementation and Evaluation Results

A. Implementation

We implemented the CoDAG based on the open source code of official Go implementation of the Ethererum. Framework of our implementation is depicted in Fig. 3, which is divided into data layer, chain layer, RPC/Console layer and Dapp layer. And the chain layer can be further divided into Network, CoDAG, Miner, and Consensus Algorithm modules as follows:

- **Network Module:** Synchronize the chain between different nodes using P2P protocols.
- **CoDAG Module:** Define the basic data structure and maintain the state of the DAG-based chain.
- **Miner Module:** Mine new blocks based on PoW mechanism and adjust the difficulties.
- **Consensus Algorithm Module:** Maintain the authorized chain based on CoDAG's algorithms.

B. Evaluation Results

1) Throughput in Different Blocksizes and Widths: Fig.4 shows the relationship between throughput and different

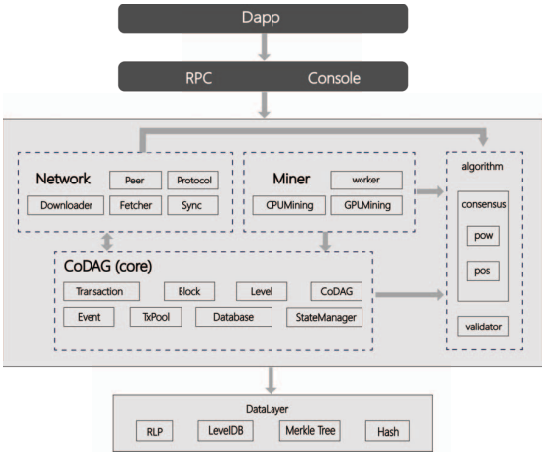


Fig. 3: Implementation Framework

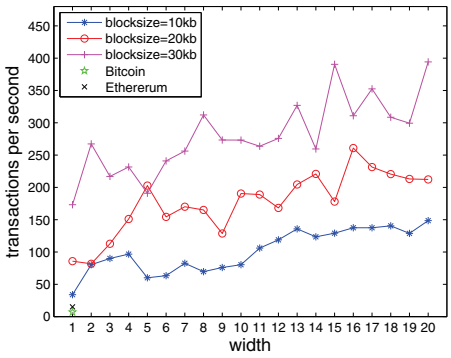


Fig. 4: Throughput of CoDAG in different blocksizes and widths

blocksizes and widths. We can observe that throughput of CoDAG scales in blocksize and width. For example, provided blocksize is 30kb, when width is 20, throughput achieves 394 tx/s, twice as much as that when width is 1. Moreover, provided width is 20, when blocksize is 30kb, throughput is 2.6 times of that when blocksize is 10kb, and 1.9 times of that when blocksize is 20kb. We also notice CoDAG always outperforms the Bitcoin and Ethererum. When blocksize is 30kb and width is 20, CoDAG achieves 56× Bitcoin's throughput and 26× Ethererum's throughput respectively. At the same time, we note there exists some fluctuations as width grows. The reasons of fluctuations are twofolds: 1) larger width means more block information, which will cause longer propagation time and degrade the throughput performance; 2) the transaction generation rate and amount are instable, which needs further improvement in our future work.

2) Throughput in Different Level Formation Time: Fig.5 illustrates the relationship between throughput and different level formation time when blocksize is 20kb. We can observe that when blocksize is 20kb and level formation time is 18s, throughput is lower than that when formation time is 8s and 13s. Meanwhile, there is no obvious throughput deviation when level formation time is 8s and 13s. For example, when width is 10, throughput shares the same value when

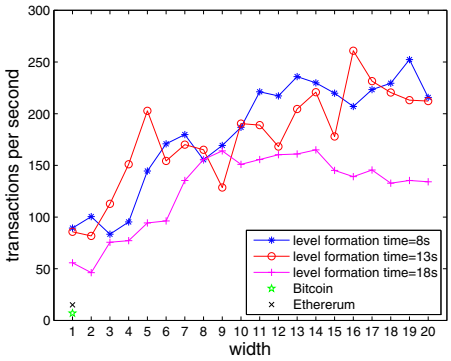


Fig. 5: Throughput of CoDAG in different level formation time when blocksize=20kb

level formation time is 8s and 13s. Theoretically, the less level formation time is, the more block information will be packaged in a unit time, thus throughput will increase. However, when formation time decreases to a small value, due to the limited capacity, nodes can not handle so many computational tasks, and those extensive transactions can not be verified in time, thus the throughput shows no obvious deviation.

VII. Conclusion and Future Work

In this paper, we propose a compacted DAG-based blockchain protocol CoDAG that greatly improves the throughput and performance of blockchain. We design a compacted DAG structure, where blocks are organized in levels and connected well within a channel. CoDAG addresses the confirmation time problem existed in previous DAG-based schemes, and it is secure enough to resist two attacks we demostrate. We implement a CoDAG prototype, and simulation results show that CoDAG's throughput is much better than Bitcoin and Ethererum.

Although CoDAG achieves a better throughput than other pure on-chain solutions, it still can not match the speed that many applications need. In the future, we will try to improve it, such as: 1) tune the parameters for better performance, 2) combine with hybrid consensus to improve the throughput further.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.
[2] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 281–310.
[3] F. R. Yu, J. Liu, Y. He, P. Si, and Y. Zhang, "Virtualization for distributed ledger technology (vdlt)," *IEEE Access*, vol. 6, pp. 25 019–25 028, 2018.
[4] S. Popov, "The tangle," https://iota.org/IOTA_Whitepaper.pdf.
[5] Y. Sompolinsky and A. Zohar, "Phantom: A scalable blockdag protocol," https://eprint.iacr.org/2018/104.pdf.
[6] Y. Sompolinsky, Y. Lewenberg, and A. Zohar, "Spectre: Serialization of proof-of-work events: Confirming transactions via recursive elections," https://eprint.iacr.org/2016/1159.pdf.
[7] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang, "Untangling blockchain: A data processing view of blockchain systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 7, pp. 1366–1385, 2018.
[8] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, 1999, pp. 173–186.
[9] J. Poon and V. Buterin, "Plasma: Scalable autonomous smart contracts," *White paper*, pp. 1–47, 2017.
[10] J. Lind, I. Eyal, P. Pietzuch, and E. G. Sirer, "Teechan: Payment channels using trusted execution environments," *arXiv preprint arXiv:1612.07766*, 2016.