Computer Security Symposium 2019
21 - 24 October 2019

BLDAG: Generalization of the Blockchain into Bi-Layered Directed Acyclic Graph

Atsuki Momose[1,a)] Jason Paul Cruz[2] Yuichi Kaji[3]

Abstract: A consensus protocol in a permission-less model is crucial in realizing cryptocurrencies and smart contracts. Nakamoto Consensus uses a Proof of Work (PoW) to form a blockchain, and it is adopted in many applications including Bitcoin and Ethereum. The blockchain is a convenient structure, but it is so simple that some inflexibility is brought to the system. As the result, problems such as the security-scalability tradeoff and wastes of energy have been pointed out for Nakamoto Consensus. Some researchers study a blockDAG instead of a blockchain, but it is not easy to reach a consensus on the total order of possibly concurrent blocks in the DAG sturucture. This study proposes a two-step hash puzzle, and extends the blocks to have two types of hash references. The hash references define a DAG structure and, additionally, an expectedly chain struture that plays essentially the same role as the blockchain of Nakamoto Consensus. A novel consensus protocol, Bi-Layered DAG (or BLDAG), is then developed on the two-step hash puzzle and the extended blocks. BLDAG inherits many favorable properties of Nakamoto Consensus, while problems such as security-scalability tradeoff and wastes of energy are avoided in BLDAG.

Keywords: blockchain, blockDAG, Proof-of-Work, scalability, cryptocurrency

1. Introduction

1.1 Background

Nakamoto Consensus[1] is widely recognized as a base component of Bitcoin, but we can regard Nakamoto Consensus as a general protocol for the state machine replication in the permission-less model (permission-less SMR)[2]. A *state machine replication* is an abstraction which bring autonomous *nodes* in a network to reach a single consensus of a state machine. The abstraction is said to be *permissionless* if nodes can participate to and leave from the network arbitrary. This means that nobody knows the current membership of the network, and each node does not know how many nodes are indeed participating to the network. Because of its simplicity and versatility, Nakamoto Consensus is used in many applications including the smart contract platform such as Ethereum[3], cryptocurrencies[4], access-control[5], [6], e-voting[7] and a business usecase in a permissioned model[8].

In Nakamoto Consensus, distributed nodes form an authenticated data structure that is called a *blockchain*[9], [10], [11]. The blockchain is the log of the state machine that is agreed by nodes, and defines the total order of *blocks*, and hence the total order of *transactions* that are recorded in blocks. For the consistency of transactions, it is desired that the blockchain does not have any *fork*. This means that concurrent creation of blocks should be prevented as far as possible, and Nakamoto Consensus utilizes *Proof-of-Work* (PoW) to control the block creation rate. The amount of "work" needed to create a block is controlled by a parameter that is called *PoW difficulty*. The PoW difficulty is adjusted so that the average interval of block creation is much more than the predetermined estimation of the maximum communication delay in the underlying P2P network. The problem in practice is that the estimation of the delay tends to be too-conservativeq if one tries to avoid security issues. The use of overestimated delay increases PoW difficulty more than actually needed, which sacrifices the transaction throughput for the expense for the increased security. This phenomena is known as the security-scalability tradeoff, and is regarded as one of most critical problems of Nakamoto Consensus.

Several efforts have been made to mitigate the security-scalability tradeoff of Nakamoto Consensus. One approach is to extend or utilize a classical solution for "permissioned" consensus[12], [13], [14] to permission-less model as in [2], [15], [16]. In these studies, a permissionless consensus (or an initial agreement) is used to select several nodes, and the *committee* that consists of those selected nodes processses transactions using a permissioned consensus protocol. This approach is advantageous to achieve the property that is called *responsiveness*, but the framework requires

[1] Graduate School of Informatics, Nagoya University
[2] Graduate School of Information Science and Technology, Osaka University
[3] Graduate School of Informatics, Nagoya University
[a)] momose@sqlab.jp

committee members to do their job for certain time of period, and spoils the permission-less nature of the network. The framewrok is not favorable from the viewpoint of security also; important decisions are made in a centralized manner by the committee which is a small subset of nodes. Smaller in number means more risk to be corrupted. Once the committee is corrupted and hijacked, then there is no way to recover the security of the system.

Some reseacrhers challenge the security-scalability tradeoff by extending the blockchain to more general structure. For example, SPECTRE[17] consider a structure that is called *blockDAG*. As the name suggests, blockDAG is a DAG (directed acyclic graph) of blocks which allows branches (forks) in the graph structure. This means that blocks can be constructed concurrently, which improves the scalability of the protocol. However, the DAG structure brings an obvious problem; it does not define a total order of blocks. If two blocks in a blockDAG are connected by a directed path, then the blocks are naturally ordered. On the other hand, blocks that do not have a directed path (sibling blocks, for example) cannot be ordered solely by blockDAG. SPECTRE tries to solve this problem by developping an algorithm to decide the order of arbitrary two blocks, but it does not determine a total order of blocks and hence not a full solution of the state machine replication problem. PHANTOM[18] is another mechanism that utilizes the blockDAG and tries to determine a total order of blocks, but it is not clear if the computational result of the algorithm is well-defined or not.

1.2 Our Approach and Contribution

In the recognition of the authors, a block in Nakamoto Consensus plays two different roles; the notarization of transactions, and the ordering of transactions. In Nakamoto Consensus, transactions are publicly notarized when they are included in a block with a valid PoW *nonce*. The creation of a block simultaneously defines the order of transactions because every block has a *reference* to a preceding block, and the references are expected to define a linear structure of the blockchain. In this framework, we need to avoid that two or more blocks are created concurrently, because such blocks bring a fork in the blockchain. Therefore, the block creation rate is intentionally suppressed in Nakamoto Consensus, which also suppresses the throughput of transactions and causes the security-scalability tradeoff.

The motivating idea of this study is to split the two roles of blocks to two different structures. For the notarization of transactions, we consider a framework in which blocks are created with relatively small PoW, and the blocks form a blockDAG instead of a blockchain. This framework contributes to scale the transaction throughput, but blocks are not ordered totally due to the DAG structure. To complement this drawback of DAG, we consider to form another structre, or a *layer*, of blocks that is used for the ordering of blocks (transactions). A key point in this construction is that not all blocks participate to the second layer of blocks. Only qualified blocks form the second layer, and contributes to define the total order of blocks.

To implement the above idea, we first introduce a *two-step hash puzzle*, which is a generalization of a hash puzzle that is commonly used in many state machine replication protocols. The two-step hash puzzle has two types of solutions, *L1*-solutions and *L2*-solution where the latter is a small subset of the former. The difference of solutions brings a difference of blocks, L1-blocks and L2-blocks, again the latter is a subset of the former. A block is also modified to have two types of references, L1-references that are used to form a DAG of L1-blocks, and an L2-refernce that is used to form a (expected) chain of L2-blocks. Everything has two layer, and we name this protocol a *Bi-Layered DAG* or *BLDAG*. The DAG of the L1-block layer constributes to fast creation of blocks, while the expected chain of L2-blocks contributes to robust convergence of the computation of the total order of blocks and transactions, and in total, the security-scalability tradeoff is resolved in BLDAG. It is also noted that BLDAG is more energy efficient than Nakamoto Consensus, and the security of BLDAG can be discussed formally through a problem reduction to the security of Nakamoto Consensus.

Basic notions of the state machine replication are introduced in Section 2. In Section 3, blockDAG and Nakamoto Consensus are briefly reviewed. The two-step hash puzzle and BLDAG are introduced in Section 4, together with related discussion.

2. Preliminary

2.1 Terms and Symbols

We consider a peer-to-peer network that consists of *nodes*, and assume that all nodes agree on a consensus protocol $\mathcal{P}$. A node is either of a *honest* node that always follows the protocol $\mathcal{P}$, or a *malicious* (or *corrupt*) node that may violate the rule of the protocol. The set of all nodes is denoted by $\mathcal{N}$, and the sets of honest and malicious nodes are denoted by $\mathcal{N}_\beta$ and $\mathcal{N}_\alpha$, respectively (we have $\mathcal{N} = \mathcal{N}_\alpha \cup \mathcal{N}_\beta$).

Each node has the ability to compute a cryptographic hash function H that maps an arbitrary input to an integer in $[0, M-1]$. The computational power differ from node to node, and we write $hp(v)$ for the number of hash computation that a node $v \in \mathcal{N}$ can perform in a unit time (in a second, for example), and call $hp(v)$ the *hash power* of v. We extend the notation and write $hp(V) = \sum_{v \in V} hp(v)$ for a set V of nodes. The *malicious fraction* of hash power is defined as $hp(\mathcal{N}_\alpha)/hp(\mathcal{N})$ and denoted by α. The *honest fraction* of hash power is defined as $hp(\mathcal{N}_\beta)/hp(\mathcal{N})$ and denoted by β.

A *hash puzzle* is a game to find a value m that makes $H(m)$ satisfy a certain condition. For general discussion, we formalize hash puzzles in terms of *oracle*. In this study, an oracle is written by a predicate with two arguments as $\mathcal{O}(d, m)$. The first argumet is called a *difficulty level* and chosen to satisfy $0 \leq d \leq M-1$. The boolean value of $\mathcal{O}(d, m)$ is randomly assigned for each m, but the assignment is controlled so that the fraction of m that makes $\mathcal{O}(d, m)$ true is $1/d$. In Nakamoto Consensus, for example,

$\mathcal{O}(d, m)$ is true if and only if $H(m) < M/d$.

The communication between nodes are made in a peer-to-peer manner, and a certain time is needed for a data to travel from one node to another. The time needed for communication is commonly called a *delay*. We write δ for the maximum delay that can occur in a considered peer-to-peer network. In practice, it is difficult to determine the actual value of δ because it depends on many factors such as the network topology, the diameter of the network, and employed communication technologies. A protocol designer may introduce a safely estimated upper bound Δ that satisfies $\delta \leq \Delta$.

2.2 Consensus Protocol

To discuss important properties of consensus protocols, several notions and aspects in [2] are briefly reviewed.

In a certain circumstance, anyone can join a peer-to-peer network with no authentication of a designated administrator. A node can come, and a node can leave arbitrarily, and nobody can say how many nodes are actually participating to the network. This is called a *permissionless* model of a network.

Consider that every honest node receives one or more transactions in each time step, and outputs a *log* which is a total order of a subset of transactions that the node has received so far. We write LOG_t^v for the log of a honest node $v \in \mathcal{N}_\beta$ at time t. The rule for the computation of logs is determined by a *consensus protocol*. A consensus protocol should accomplish complicated tasks, and several properties of consensus protocols have been proposed and discussed. In a general framework of distributed programming, those properties can be categorized to either of the *safety* or the *liveness*[19]. The safety is the category of properties that must be satisfied for all the time. A violation to the property in this category cannot be recovered, and brings fatal disruption of the protocol. The liveness (in a broader sense) is the category of properties that should be realized at a certain time in the future. With focusing on the security aspects of distributed programming and consensus protocols, we discuss the *consistency* which is categorized to the safety, and the *liveness* (in a narrow sense) which is categorized to the (broader) liveness.

Definition 1 (Consistency)
A consensus protocol $\mathcal{P}$ has *consistency* if the following two conditions are satisfied.

(1) For honest nodes v and v', and times t and t', either $LOG_t^v \prec LOG_{t'}^{v'}$ or $LOG_{t'}^{v'} \prec LOG_t^v$ holds, where $A \prec B$ means that A is a prefix of B.

(2) For a honest node v and times t and t' with $t < t'$, we always have $LOG_t^v \prec LOG_{t'}^v$.

Intuitively, the first condition means that there is no disagreement of the the total order of transactions among honest nodes. The second condition means that a log is never modified once it is determined.

Definition 2 (Liveness)
A consensus protocol $\mathcal{P}$ has *liveness* if there is a constant T_{confirm} such that, for any time t, a transaction that is given to any of honest nodes is included in $LOG_{t'}^v$ for every honest node $v \in \mathcal{N}_\beta$ and every time t' with $t' > t + T_{\text{confirm}}$.

This means that a transaction propagates to all nodes and confirmed after certain confirmation time T_{confirm}.

Definition 3 (Responsiveness)
A consensus protocol $\mathcal{P}$ with the liveness property is said to be *responsive* if the confirmation time T_{confirm} depends only on the actual maximum delay δ of the network, (i.e. it does not depend on the upper bound Δ that has been chosen by a protocol designer).

3. Graphs of blocks

3.1 BlockDAG

For a general discussion, we define a *block* as a data object that has *hash references* (or simply *references*) to other blocks. A block with no hash reference is designated as a *genesis* block. Consider a set G of blocks whose hash references are closed in G, and G can be regarded as a directed graph where an edge from $b_1 \in G$ to $b_2 \in G$ is defined if and only if b_1 has a hash reference to b_2. The set G is called a *blockDAG* if G defines a directed acyclic graph. For a block $b \in G$, we write $past(b)$ for the set of blocks that are reachable from b with the directed edges in G, and $tips(G)$ for the set of blocks to which no block has hash references. In other words, $past(b)$ is the set of ancestors of b and $tips(G)$ is the set of leaves of the DAG.

In the consensus protocols SPECTRE[17] and PHANTOM[18], a blockDAG is formed and maintained by nodes in the peer-to-peer network. For a node $v \in \mathcal{N}$ and a time t, we write G_t^v for the blockDAG that is possessed by v at time t. The blockDAG of each node is updated through the operations of *mining* and *propagation*.

- In a mining, a node v tries to create a new block b that contains fresh transactions (or the root of the Merkle hash tree of transactions), hash references to all blocks in $tips(G_t^v)$, and a PoW nonce. The PoW nonce r must be selected so that it makes $\mathcal{O}(d, b(r))$ true, where d is the PoW difficulty that is agreed by all nodes, and $b(r)$ is the block with a nonce r. It is noted that $past(b(r)) = G_t^v$ because b has hash references to all blocks in $tips(G_t^v)$. The block $b(r)$, if successfully created, is appended to the blockDAG G_t^v of the node v, and transmitted to all neighbor nodes in the peer-to-peer network.
- In a propagation, a node v receives a block b from its neighbor node, and verifies that b is not included in G_t^v and that $\mathcal{O}(d, b)$ is true. If two conditions hold, then the node v appends the block b to its blockDAG G_t^v, and transmits b to all neighbor nodes in the peer-to-peer network. A block propagates to all honest nodes within the maximum delay δ, and therefore $G_t^v \subset G_{t+\delta}^{v'}$

holds for any $v, v' \in \mathcal{N}_\beta$.

3.2 Nakamoto Consensus

In Nakamoto Consensus[1], a block has only one hash reference, and the PoW difficulty is controlled so that two or more blocks are hardly created concurrently. In this setting, a newly created block is expected to propagate to all honest nodes before somebody else creates another block. In the most ideal scenario, the directed graph that is defined by the blocks becomes a chain that is called a *blockchain*, and $tips(G)$ contains only one block that is the "tail" of the blockchain. In practice, however, there is chance that two or more blocks are created concurrently. If this happens unfortunately, then a *fork* is brought to the blockchain and the structure becomes a tree. In case there is a fork in the blockchain, honest nodes select the series of blocks that lead to the longest chain. The mining operation is then explained as follows.

- In a mining, a node v tries to create a new block b that contains fresh transactions, a hash references to the tail of the blockchain, and a PoW nonce. The PoW nonce r must be selected so that it makes $\mathcal{O}(d, b(r))$ true, where d is the PoW difficulty that is agreed by all nodes. Noted that $past(b(r)) = G_t^v$. The block $b(r)$, if successfully created, is appended to the blockchain G_t^v of the node v, and transmitted to all neighbor nodes in the peer-to-peer network.

The propagation of blocks are made similarly to the blockDAG case.

Different from blockDAG, all blocks are naturally ordered in G_t^v as a blockchain. One small caution is that G_t^v is the information that is local to the node v at time t. The node v should remind that there can be some other blocks that are not yet delivered to v, and some tail blocks in the blockchain might be pushed out from the longest chain. For the sake of safety, we should keep s tail blocks in the blockchain in "pending"; the total order of blocks and transactions in these s pending blocks are delayled until they get sufficiently aged.

We write Nakamoto Consensus with difficulty level d and pending margin s by $\mathcal{P}_{nak}(d, s)$.

4. Bi-Layered DAG

4.1 Two-Step Hash Puzzle

In a PoW-based consensus protocol, the mining operation requires a search of a nonce that makes an oracle true. The oracle is commonly implemented by a cryptographic hash function, and the nonce is regarded as a solution of a certain hash puzzle. The rate of block creation is controlled by adjusting the difficulty level of the hash puzzle, but it is not good idea to control everything with only one parameter. To bring a better trade-off relation in a consensus protocol, we consider a *two-step hash puzzle* that can have two different difficulty levels.

Definition 4 (two-step hash puzzle)
A two-step hash puzzle is defined by two independent oracles $\mathcal{O}_1$ and $\mathcal{O}_2$ with two difficulty levels d_1 and d_2. Write a block that is equipped with a nonce r by $b(r)$. A nonce r is said to be a L1-solution *if it makes $\mathcal{O}_1(d_1, b(r))$ true, and a* L2-solution *if it makes both of $\mathcal{O}_1(d_1, b(r))$ and $\mathcal{O}_2(d_2, b(r))$ true. A block is an* L1-block *(resp.* L2-block*) if the nonce of the block is an L1-solution (resp. L2-solution).*

In practice, the oracles can be implemented by using a cryptographic hash function H with range $[0, M-1]$: $\mathcal{O}_1(d_1, b)$ is true if and only if $H(b) < M/d_1$ ($H(b)$ is considered as a binary integer), and $\mathcal{O}_2(d_2, b)$ is true if and only if $H(H(b)) < M/d_2$. A randomly chosen nonce is a L1-solution with probability $1/d_1$, and a L2-solution with probability $1/(d_1 d_2)$. We will choose difficulty levels d_1 and d_2 so that it is not very difficult to find an L1-solution, but it is extremely difficult to find an L2-solution (thus it is unlikely that L2-blocks are created concurrently).

4.2 BLDAG

4.2.1 Basic Idea

A blockDAG is advantageous to scale transaction process, but there is difficulty in handling blocks that are incomparable in the DAG structure. To overcome the difficulty, the authors consider to use L2-blocks as "milestones" in the DAG structure, and sort L1-blocks to groups that are separated by milestone L2-blocks. The computation of the total order is closed in each group, which leads fast convergence of the computation. The creation of milestone L2-blocks is controled in such a way that the L2-blocks form a virtual blockchain that is embedded in the DAG of blocks. In other words, the DAG of blocks has another layer of a chain of blocks, and we call this structure a *Bi-Layered DAG* or *BLDAG*. To realize BLDAG, we let blocks have two types hash references, and utilize the two-step hash puzzle in the mining procedure.

4.2.2 Blocks in BLDAG

Blocks in BLDAG have two types of hash references. References of the first type are called *L1-references* or *DAG references* and used to point L1-blocks. References of the second type are called *L2-references* or *chain references* and used to point L2-blocks. A block has one or more L1-references and only one L2-reference.

Honest nodes in the network possesses a set of blocks, and the set is updated through the operations of mining and propagation. We write G_t^v for the set of blocks that a node v possesses at time t. With the L1-references (DAG references) of the blocks, G_t^v defines a DAG of blocks, which we call an *L1-DAG*. We write $past_1(b)$ for the set of blocks that are reachable from a block b in the L1-DAG, and $tips_1(G_t^v)$ for the set of leaf blocks in the L1-DAG.

Besides L1-DAG, G_t^v defines another graph structure by regarding L2-references (chain references) as directed edges. The *L2-tree* that is induced from G_t^v consists of the set of L2-blocks in G_t^v and directed edges that are defined from

L2-references in the L2-blocks. The graph is indeed a tree because a block has only one L2-reference. For a block b, we write $past_2(b)$ for the set of L2-blocks that are reachable from b with L2-references only, and $tips_2(G_t^v)$ for the set of L2-blocks to which no L2-block has an L2-reference. In other words, $past_2(b)$ is the set of ancestors of b in the L2-tree, and $tips_2(G_t^v)$ is the set of leaves of the L2-tree. A *chain* in an L2-tree is a sequence of L2-blocks $[b_1, \ldots, b_l]$ such that b_1 is the root of the L2-tree, and b_i with $i > 1$ has an L2-reference to b_{i-1}. The last block b_l in the chain is the *tail* of the chain, and l is the *length* of the chain. A chain with the largest length is called the *longest* chain.

We will manage so that the L2-tree rarely has a fork, and prepare a procedure that utilizes the "longest-chain principle" of Nakamoto Consensus to determine the trunk of the tree in case a fork is created by an accident.

4.2.3 Mining in BLDAG

The mining protocol of BLDAG has both flavors of SPECTRE and Nakamoto Consensus. In the preparation of a new block, a node (miner) v includes the block with fresh transactions (or the root of the Merkle has tree of transactions), L1-references to all blocks in $tips_1(G_t^v)$, and an L2-reference to the tail of the longest chain in the L2-tree. When L2-tree has no fork, $tips_2(G_t^v)$ contains only one L2-block and that block is the tail of the longest chain. If $tips_2(G_t^v)$ contains two or more L2-blocks, then the node v chooses the one that leads the longest chain in the L2-tree. For the prepared block b, the node v tries to search a PoW nonce r that is an L1- or L2-solution of the two-step hash puzzle. If the node v succeeds to find a solution r, then v includes the block $b(r)$ in G_t^v, and propagate $b(r)$ to all neighbor nodes in the peer-to-peer network.

A crucial point in the above algorithm is that a node succeeds to create a new block if it can find an L1-solution r of the two-step hash puzzle. By definition, the search of L1-solutions is easier than the search of L2-solutions. This allows quick creation of L1-blocks, and contributes to scale the transaction processing. The newly created L1-block $b(r)$ is included in the L1-DAG, but $b(r)$ is not included in the L2-tree unless the PoW nonce r happens to be an L2-solution of the two-step hash puzzle. Because L2-solutions are difficult to find, it is unlikely that two or more L2-blocks are created concurrently or in a short period of time, which means that the L2-tree rarely has a fork, and thus the tree is in a chain-like form.

Figure 1 illustrates L1-references and L2-references of a certain set of blocks. In the figure, L1-blocks are shown in rectangles and L2-blocks are shown in pentagons. A block has multiple L1-references to L1-blocks that were in $tips_1(G_t^v)$, and the L1-references define a DAG structure. On the other hand, a block has only one L2-reference to the tail L2-block of the L2-tree, and L2-references define a tree in general. Delete L1-blocks from the tree, and we obtain the L2-tree that consists of L2-blocks only. In this example, The L2-tree has only three nodes; gen, d and l, which forms a chain of length three. In the next mining, a new block

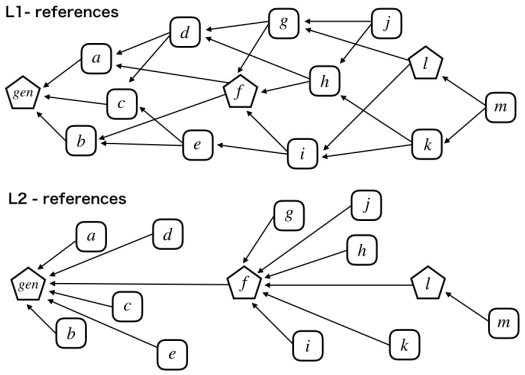

Fig. 1 Bi-Layered DAG

should have L1-references to blocks j and m which are in $tips_1(G_t^v)$, and an L2-reference to the block l which is the tail of the L2-tree.

4.2.4 Ordering Protocol

The L1-DAG defines a topological order of blocks, but it is a partial order and inadequate for the consensus protocol. We make use of L2-blocks in the L1-DAG to separate subsets of blocks, and introduce some additional rules to define a total order of blocks. In this section, the order of blocks are written $b_1 \lhd b_2$ for simplicity.

Basically, the order $\lhd$ is determined according to four principles. The first principle is that the order $\lhd$ is compatible with the topological order that is defined by the L1-DAG. This means that if $b \in past_1(b')$ for blocks b and b', then we must have $b \lhd b'$. The second principle of the order $\lhd$ is that an L1-block cannot "jump over" incomparable L2-blocks. To explain this principle, consider a chain $[b_1, \ldots, b_l]$ of L2-blocks in the L2-tree. We must have $past_1(b_{i-1}) \subset past_1(b_i)$ by definition, and we can define $\mathit{diff}_1 = past_1(b_1) = \emptyset$ (because b_1 is the genesis block) and $\mathit{diff}_i = past_1(b_i) - past_1(b_{i-1})$ for $i \geq 2$. In constructing the order $\lhd$, we require that

$$b_{i-1} \lhd (\text{blocks in } \mathit{diff}_i) \lhd b_i \qquad (i \geq 2).$$

The above two principles are not sufficient to determine the order among blocks in diff_i. To tackle this problem, the third principle is utilized: a block in diff_i is given a preceding position in the order $\lhd$ if it has more descendants in diff_i. For an L2-block b_i in the chain and an L1-block $b \in \mathit{diff}_i$, define $P[b, b_i]$ as the number of blocks $b' \in \mathit{diff}_i$ satisfying $b \in past_1(b')$ (thus $P[b, b_i]$ is the number of descendants of b within the set diff_i). For blocks $b, b' \in \mathit{diff}_i$, we let $b \lhd b'$ if $P[b, b_i] > P[b', b_i]$. In case $P[b, b_i] = P[b', b_i]$ for $b, b' \in \mathit{diff}_i$, we let $b \lhd b'$ if $H(b) > H(b')$ as the final fourth principle, where hash values are regarded as binary integers.

The above described principles are described as Algorithm 1 that determines an ordered list Λ of blocks. In the algorithm, we keep s blocks in the end of the longest chain from participating to the computation, where s is a safety mar-

gin that is determined by a system designer, because there is small risk that those fresh s blocks get pushed out from the longest chain (see the section of Nakamoto Consensus).

Algorithm 1 OrderDAG

Input: G_t^v - the set of blocks that a node v possesses at time t
Output: Λ - an ordered list of blocks in G_t^v

1: construct a chain $\gamma = [b_1, \ldots, b_l]$ of L2-blocks by trimming the last s L2-blocks from the longest chain in the L2-tree.
2: initialize $\Lambda = \emptyset$
3: **for** $i = 1, 2, \ldots, l$ **do**
4: sort all blocks in $diff_i$ in the descending order and construct a list λ_i of blocks in $diff_i$, where the sorting is made according to the main-key $P[b, b_i]$ and the sub-key $H(b)$ for a block b.
5: append λ_i and b_i to Λ
6: **return** Λ

To illustrate the computation in Algorithm 1, remind the example in Figure 1. For simplicity, we do not trim the tail of L2-blocks, namely we assume $s = 0$. In this case, the chain $\gamma = [gen, f, l]$, and $diff_1 = \emptyset$, $diff_2 = \{a, b\}$, and $diff_3 = \{c, d, e, g, i\}$. To order blocks in $diff_i$, assume here for simplicity that a block with the smaller alphabet has the smaller hash value. As a result of the computation, the final order that is determined by the algorithm is $\Lambda = [gen, a, b, f, c, d, e, g, i, l]$. The remaining blocks $\{h, j, k, m\}$ are not included in the order at this moment since they are not referenced by any L2-blocks yet.

We write BLDAG with difficulty level (d_1, d_2) and pending margin s by $\mathcal{P}_{bld}(d_1, d_2, s)$.

4.3 Evaluation

4.3.1 Security

A consensus protocol is designed to satisfy several requirements, and it is not appropriate to discuss its security based on a single criteria. It is important to investigate a protocol from diffirent viewpoints, and the *consistency* and the *liveness* that have been defined in Definitions 1 and 2 are especially significant for that sake. Intuitively saying, the consistency is the property that the order of blocks (transactions) that has been agreeded by honest nodes cannot be overwritten by malicious nodes, and the liveness is the property that transactions are processed in some reasonable time. It is known that Nakamoto Consensus has these propoerties under some environments [20],[21],[22], and we show that BLDAG also possesses the properties.

Lemma 1 (consistency of BLDAG)
If $\mathcal{P}_{nak}(d_1 d_2, s)$, Nakamoto Consensus with difficulty level $d_1 d_2$ and pending margin s has consistency, then $\mathcal{P}_{bld}(d_1, d_2, s)$, BLDAG with difficulty level (d_1, d_2) and pending margin s also has consistency.

Proof The notion of consistency has been originally defined for transactions in Definition 1, but it can be extended to blocks in a natural manner because a block defines an ordered list of transactions. The consistency with respect to blocks is called a *block consistency* in this proof. We can also regard consensus protocols output total orders of blocks rather than transactions. Consensus protocols viewed in this regard are called *block consensus protocols*, and we can consider a block Nakamoto Consensus protocol $\mathcal{P}_{Bnak}$ and a block BLDAG $\mathcal{P}_{Bbld}$. Due to obvious correspondence, we can show that if $\mathcal{P}_{nak}(d_1 d_2, s)$ has consistency, then $\mathcal{P}_{Bnak}(d_1 d_2, s)$ has block consistency.

The block consistency of $\mathcal{P}_{Bnak}(d_1 d_2, s)$ brings the block consistency of L2-blocks in $\mathcal{P}_{Bbld}$. Remind that an L2-block in BLDAG is a data object that has a single hash reference to another L2-block, and the creation of an L2-block is accomplished if an L2-solution of a two-step hash puzzle is found. The discovery of an L2-solution of a two-step hash puzzle can be regarded as a discovery of a solution to the usual hash puzzle with difficulty level $d_1 d_2$, and the creation of L2-blocks in block BLDAG and the creation of blocks in block Nakamoto Consensus can be regarded as the same stochastic process. This guarantees that the if $\mathcal{P}_{Bnak}(d_1 d_2, s)$ has block consistency, then $\mathcal{P}_{Bbld}(d_1, d_2, s)$ also has block consistency for L2-blocks. In block BLDAG, the order of blocks in the set $diff_i$, and the order between blocks in $diff_i$ and L2-blocks are uniquely determined (see Algorithm 1). Therefore, if block consistency is guaranteed for L2-blocks, then the property also applies to all L1-blocks in block BLDAG. Because transactions are contained in blocks with totally oredered manner, the block consistency of $\mathcal{P}_{Bbld}(d_1, d_2, s)$ implies the consistency of $\mathcal{P}_{bld}(d_1, d_2, s)$. □

Lemma 2 (liveness of BLDAG)
If $\mathcal{P}_{nak}(d_1 d_2, s)$, Nakamoto Consensus with difficulty level $d_1 d_2$ and pending margin s has liveness, then $\mathcal{P}_{bld}(d_1, d_2, s)$, BLDAG with difficulty level (d_1, d_2) and pending margin s also has liveness.

Proof If $\mathcal{P}_{nak}(d_1 d_2, s)$ has liveness, then, a transaction x that is given to a honest node propagates to the network and is included in a block that is created by a certain honest node. It is derived immediately that at least one block is created by honest nodes at every time period T_{confirm} in $\mathcal{P}_{Bnak}(d_1 d_2, s)$, which can be replaced with the context of L2-blocks in $\mathcal{P}_{Bbld}(d_1, d_2, s)$. When a honest node creates a L2-block after the transaction x has been given to any of honest nodes, there are two cases; the transaction x has been included in a certain L1-block, or there is no block that includes x. If x has been included in an L1-block, then the L1-block has been appended to the L1-DAG and referenced by the L2-block. At this moment, the transaction x is included in the log that is defined from the total order of blocks. As for the second case, if there is no block that includes x, then the L2-block must include x. All transactions in the L2-block are included in the log immediately, and in either cases, the transaction x is included in the log. The duration to the creation of the L2-block is controlled by the difficulty level $d_1 d_2$. If the duration in $\mathcal{P}_{nak}(d_1 d_2, s)$ is T_{confirm}, then the duration in $\mathcal{P}_{bld}(d_1, d_2, s)$ is T_{confirm} also. This proves the liveness of $\mathcal{P}_{bld}(d_1, d_2, s)$. □

4.4 Scalability

In theory, an arbitrary number of transactions can be included in a single block. In practice, however, the number of transactions in a block is limited by a certain constant which we denote by k. Therefore the number of transactions that can be processed in a unit time equals to k times the number of blocks that are created in a unit time. The total hash power in the peer-to-peer network is $hp(\mathcal{N})$, and the expected number of blocks that are created in a unit time is obtained by dividing $hp(\mathcal{N})$ by a difficulty level of the hash puzzle. In Nakamoto Consensus with difficulty level d, the number of transactions that can be processed in a unit time is

$$\sigma = k\frac{hp(\mathcal{N})}{d}.$$

In BLDAG, this number is given as

$$\sigma' = k\frac{hp(\mathcal{N})}{d_1}.$$

where d_1 is the difficulty level for the L1-solution of the two-step hash puzzle. The two equations seem the same, but the two consensus protocols have different constraints for the choice of the difficulty levels.

In Nakamoto Consensus, the difficulty level d is adjusted so that the interval of block creation is expectedly greater than Δ, which is a predetermined constant upperbound of the maximum network delay of the peer-to-peer network. Roughly saying, there is correlation between d and Δ. Using large Δ contributes to the security because a fork in a blockchain is less likely to occur with large Δ, but such a choice of Δ increases the difficulty level d and thus degrades the transaction processing rate. This is the security-scalability tradeoff of Nakamoto Consensus.

In BLDAG, a fork in the L2-tree should be avoided for the same reason, and the difficulty level d_1d_2 for creating L2-blocks should be adjusted in a similar manner to Nakamoto Consensus. Because we have two parameters, it is possible to choose d_1 small while keeping the product d_1d_2 sufficiently large. This means that the transaction processing rate is not sacrificed for the security, and both of the security and scalability are achieved in BLDAG.

4.5 Energy Efficiency

In Nakamoto Consensus, many nodes (miners) compete for creating a new block but there is only one winner. Once a winner is decided, all the efforts and resources that were consumed by other miners come to nought. With the wide spread of cryptcurrencies, quite a lot energy is needed to solve a hash puzzle today. This means that vast amounts of energy is being wasted just to decide a winner in the mining competition but nothing else. In BLDAG, it is allowed that multiple L1-blocks are created concurrently. The energy that was used to solve a two-step hash puzzle contributes to create a new L1-block and thus contributes to the transaction process. In this sense, BLDAG is much prefered for the aspect of greener energy efficiency. This feature motivates more nodes to participate to the mining process, because their efforts are likely to be rewarded even if they did not participate to a big mining pool. Such a tendency increases the total hash power in the peer-to-peer network, and also increases honest fraction of hash power, making the protocol more robust against the control of malicious nodes.

4.6 Confirmation Time

Another performance concern in consensus protocols is the latency of the confirmation of transactions. In Nakamoto Consensus, all blocks are ordered on a blockchain. This means that a transaction is given a position in the total order immediately it is included in a certain block. On the other hand, newly created L1-blocks, and transactions that are included in those L1-blocks, are not ordered promptly in BLDAG. A new L1-block b is placed in the topological order of the L1-DAG, but the total order with respect to b is not determined until b has an L2-block descedant. Since the creation rate of L2-blocks in BLDAG is almost the same as the block creation rate in Nakamoto Consensus, the latency of the confirmation of transactions is expected to be the same between two protocols. Similar to Nakamoto Consensus, it is conjectured that BLDAG is not resonpsiveness (Definition 3).

4.7 Relation to Other Consensus Protocols

BLDAG can be considered as a generalization of Nakamoto Consensus in several aspects;

(1) the hash references of blocks are augmented,
(2) two layers are introduced for blocks and references, and
(3) the ordering algorithm is refined based on the chain-like structure of L2-blocks.

Conversely, Nakamoto Consensus can be regarded as a special (or constrained) BLDAG in which a block is allowed to have only one hash reference, there is no distinction of layers, and thus there is no need of a sophisticated ordering algorithm. The constraints make the protocol simpler, but the simplicity brings some inflexibility including the security-scalability tradeoff. The blockDAG-based protocols such as SPECTRE and PHANTOM try to generalize Nakamoto Consensus by solely augmenting hash references of blocks, but a DAG is a too general structure and we face to difficulty to define a total order of blocks and transactions. In this study, we avoided the problem by introducing L2-blocks, and let L2-blocks constitute a chain-like structure in the DAG. The L2-blocks are helpful to determine the total order of other L1-blocks and transactions included in them, and make an essential contribution to the well-defined ordering algorithm.

We conjecture that the ordering alghorithm is versatile, and can be used in a DAG in which a chain-like structure is defined by a certain means. Consider for example a partially generalized Nakamoto Consenus in which hash references are augmented but there is no distinction of L1 and L2 blocks. The hash references of blocks define a chain structure mostly, but there may exist some forks in the chain as in the original Nakamoto Consensus. In the original Nakamoto Consensus,

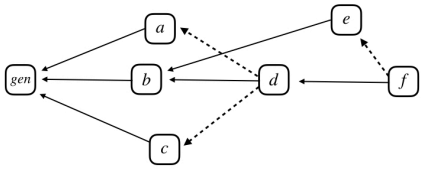

Fig. 2 partial generalization

a block is marked "loser" and ignored if the block gets off from the longest chain. On the other hand, the loser blocks have the chance to be refered by trailing blocks (see Figure 2) in the above partially generalized protocol, and given a position in the total order of blocks by the ordering algorothm. We note that this partially genralized Nakamoto Consensus is just to illustrate the contribution of our investigation. The energy efficiency of Nakamoto Consensus can be improved in this partial generalization, but it does not fully scale because we have only one difficulty level.

5. Concluding Remark

A novel consensus protocol BLDAG (Bi-Layered DAG) is presented in this study. BLDAG has no security-scalability tradeoff which has been a big obstacle in using Nakamoto Consensus in a practical, open and globally distributed circumstances. Besides the absense of the tradeoff, BLDAG is favorable from the viewpoint of energy efficiency. All the energy that is used to solve a hash puzzle fuels the system, and contributes to process transactions. This motivates more nodes to participate to mining, which increases the total hash power of the network and makes the system robust against malicious nodes. The study also investigates the security proof of BLDAG. The consistency and the liveness of BLDAG are reduced to those of Nakamoto Consensus, which have been proven in literature.

BLDAG inherits many aspects from Nakamoto Consensus, but that means that some of unfavorable properties of Nakamoto Consensus still remain in BLDAG. Consider for example the problem of *selfish mining*[23], in which a malicious node withhold a mined block in a short period of time before delivering it to the network. With selfish minining, the malicious node can make a "flying start" for investigating the next block, allowing the node have more control of the system than its actual hash power. The selfish mining is known to be a problem of fairness in Nakamoto Consensus, and a problem of the same kind may arise in the L2-tree of BLDAG. Despite this issue, the authors conjecture that the mechanism of L1-DAG may deter the attack. If a block is withhold by a malicious node, then the block loses the opportunity to be refered from other blocks. This brings some disadvantage to the block because having more descendants gives the block more significant position in the system. The authors will addressed this conjecture more in detail in the future study.

References

[1] Nakamoto, S. et al.: Bitcoin: A peer-to-peer electronic cash system (2008).

[2] Pass, R. and Shi, E.: Hybrid consensus: Efficient consensus in the permissionless model, *31st International Symposium on Distributed Computing (DISC 2017)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2017).

[3] Wood, G. et al.: Ethereum: A secure decentralised generalised transaction ledger, *Ethereum project yellow paper*, Vol. 151, No. 2014, pp. 1–32 (2014).

[4] Team, Z. et al.: The ZILLIQA Technical Whitepaper, *Retrieved September*, Vol. 16, p. 2019 (2017).

[5] Cruz, J. P. and Kaji, Y.: The bitcoin network as platform for trans-organizational attribute authentication (2015).

[6] Cruz, J. P., Kaji, Y. and Yanai, N.: RBAC-SC: Role-based access control using smart contract, *IEEE Access*, Vol. 6, pp. 12240–12251 (2018).

[7] Hanifatunnisa, R. and Rahardjo, B.: Blockchain based e-voting recording system design, *2017 11th International Conference on Telecommunication Systems Services and Applications (TSSA)*, IEEE, pp. 1–6 (2017).

[8] Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y. et al.: Hyperledger fabric: a distributed operating system for permissioned blockchains, *Proceedings of the Thirteenth EuroSys Conference*, ACM, p. 30 (2018).

[9] Devanbu, P., Gertz, M., Martel, C. and Stubblebine, S. G.: Authentic third-party data publication, *Data and Application Security*, Springer, pp. 101–112 (2002).

[10] Naor, M. and Nissim, K.: Certificate revocation and certificate update, *IEEE Journal on selected areas in communications*, Vol. 18, No. 4, pp. 561–570 (2000).

[11] Tamassia, R.: Authenticated data structures, *European symposium on algorithms*, Springer, pp. 2–5 (2003).

[12] Castro, M., Liskov, B. et al.: Practical Byzantine fault tolerance, *OSDI*, Vol. 99, No. 1999, pp. 173–186 (1999).

[13] Dwork, C., Lynch, N. and Stockmeyer, L.: Consensus in the presence of partial synchrony, *Journal of the ACM (JACM)*, Vol. 35, No. 2, pp. 288–323 (1988).

[14] Martin, J.-P. and Alvisi, L.: Fast byzantine consensus, *IEEE Transactions on Dependable and Secure Computing*, Vol. 3, No. 3, pp. 202–215 (2006).

[15] Abraham, I., Malkhi, D., Nayak, K., Ren, L. and Spiegelman, A.: Solidus: An incentive-compatible cryptocurrency based on permissionless byzantine consensus, *CoRR, abs/1612.02916* (2016).

[16] Liu, Z., Tang, S., Chow, S. S., Liu, Z. and Long, Y.: Fork-free hybrid consensus with flexible proof-of-activity, *Future Generation Computer Systems*, Vol. 96, pp. 515–524 (2019).

[17] Sompolinsky, Y., Lewenberg, Y. and Zohar, A.: SPECTRE: A Fast and Scalable Cryptocurrency Protocol., *IACR Cryptology ePrint Archive*, Vol. 2016, p. 1159 (2016).

[18] Sompolinsky, Y. and Zohar, A.: PHANTOM, *IACR Cryptology ePrint Archive, Report 2018/104* (2018).

[19] Cachin, C., Guerraoui, R. and Rodrigues, L.: *Introduction to Reliable and Secure Distributed Programming*, Springer (2011).

[20] Pass, R., Seeman, L. and Shelat, A.: Analysis of the blockchain protocol in asynchronous networks, *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, pp. 643–673 (2017).

[21] Garay, J., Kiayias, A. and Leonardos, N.: The bitcoin backbone protocol: Analysis and applications, *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, pp. 281–310 (2015).

[22] Rosenfeld, M.: Analysis of hashrate-based double spending, *arXiv preprint arXiv:1402.2009* (2014).

[23] Eyal, I. and Sirer, E. G.: Majority is not enough: Bitcoin mining is vulnerable, *Communications of the ACM*, Vol. 61, No. 7, pp. 95–102 (2018).