LightChain: A DHT-based Blockchain for Resource Constrained Environments

Yahya Hassanzadeh-Nazarabadi, Alptekin Küpçü, and Öznur Özkasap
Department of Computer Engineering, Koç University, İstanbul, Turkey
{yhassanzadeh13, akupcu, oozkasap}@ku.edu.tr

March 31, 2019

Abstract

As an append-only distributed database, blockchain is utilized in a vast variety of applications including the cryptocurrency and Internet-of-Things (IoT). The existing blockchain solutions have downsides in communication and storage efficiency, convergence to centralization, and consistency problems. In this paper, we propose *LightChain*, which is the first blockchain architecture that operates over a Distributed Hash Table (DHT) of participating peers. *LightChain* is a permissionless blockchain that provides addressable blocks and transactions within the network, which makes them efficiently accessible by all the peers. Each block and transaction is replicated within the DHT of peers and is retrieved in an on-demand manner. Hence, peers in *LightChain* are not required to retrieve or keep the entire blockchain. *LightChain* is fair as all of the participating peers have a uniform chance of being involved in the consensus regardless of their influence such as hashing power or stake. *LightChain* provides a deterministic fork-resolving strategy as well as a blacklisting mechanism, and it is secure against colluding adversarial peers attacking the availability and integrity of the system. We provide mathematical analysis and experimental results on scenarios involving $10K$ nodes to demonstrate the security and fairness of *LightChain*.

1 Introduction

Blockchain [1] is an append-only distributed database that provides a partial ordering of blocks among a set of trust-less peers. Each block consists of a set of transactions. In a blockchain, the blocks are connected to each other via immutable links from each block to its previous one and form a chain, which is called the *ledger*. Because they define a partial ordering of blocks without the need of a global synchronized clock, provide a tamper-proof architecture, and establish trust over a trust-less system of independent peers, the blockchain systems are employed in many decentralized applications including the cryptocurrencies [1], Internet-of-Things [2, 3], digital rights management [4], big data

[5], search engines [6], fair data exchange [7], supply-chain management [8], and namespace management [9].

A blockchain system is usually modeled as a stack of protocols with at least four layers, from bottom to top are named as *Network, Consensus, Storage, and View* [10]. The layers work interoperably with each other in a pipelined manner i.e., the output of the lower layer is the input to the upper one. The Network layer deals with the dissemination mechanism of the transactions and blocks among the peers of the system. The Consensus layer represents the protocols for block generation decision-making process, which aim at providing an accepted ordering of the blocks among the peers. In other words, all the peers that follow the protocols provided by the Consensus layer are aimed to reach the same state of the generated blocks ordering. The Storage layer provides the read functionality for the peers to read from the blockchain. The View layer represents the most recent state of the participating peers' data considering all the updates on the ledger from the very first to the most recent blocks.

Existing blockchains' deficiencies: The existing blockchain solutions have scalability problems in all layers of the blockchain protocol stack. To the best of our knowledge, at the Network layer, all the existing blockchains operate on unstructured overlays [1, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28]. Such overlays have no deterministic, well-defined, and efficient lookup mechanism to retrieve the address of the peers, the content of the blocks, and the new transactions. Rather, the knowledge of a peer (i.e., other peers, blocks, and transactions) is gained by the epidemic message dissemination among the peers (e.g., broadcasting in Bitcoin [1]) with the communication complexity of $O(n)$ to disseminate a new block or transaction, where n is the number of participating peers in the system. In this paper, by the communication complexity we mean the number of the exchanged messages (i.e., the round complexity).

At the Consensus layer, the existing solutions converge to centralization by delegating the block generation decision making to a biased subset of the special peers, e.g., the peers with higher computational power [1, 23, 24, 25], higher stakes [12, 13], or longer activity history in the system [16]. Such centralization convergence allows a subset of the peers to leverage the blockchain to their advantage by, for example, performing selfish-mining [29]. The existing blockchains are also prone to the consistency problems that are caused by their probabilistic fork-resolving approach at the Consensus layer, i.e., following the longest chain of the forks as the main chain [1]. The probabilistic nature of this fork-resolving approach is due to the volatility of the main chain. This threatens the consistency and performance of the system since once the main chain is conquered by another chain, all the blocks that have been already appended to it are considered invaluable [30]. Hence, the probabilistic fork-resolving strategy causes probabilistic finalization on the block generation, i.e., the more blocks are coming after a certain block on the ledger, that chain of blocks gets longer, and with a higher probability that block is being finalized as the main chain's block. Thus, in the existing blockchain solutions, appending a generated block to the ledger does not make it effective unless a number of new blocks come

after it [31].

Having b blocks in the system, the existing blockchains require the Storage layer memory complexity of $O(b)$ by downloading and keeping the entire ledger locally at the peer's storage [10]. In other words, as peers are not able to efficiently lookup any information within the unstructured overlay, they locally store the perceived information and gradually construct a local copy of the entire ledger, which takes $O(b)$ storage complexity. Likewise, upon joining the system, during the bootstrapping phase, a new peer needs to verify the entire state of the ledger from the very first block to the most recent one to check the integrity of the ledger [30]. This imposes a time and communication complexity of $O(b)$ at the View layer. Bootstrapping is defined as the process in which a new node constructs its view of the blockchain [10].

Sharding: The best existing approach to overcome the mentioned performance and scalability problems of the blockchains is to apply sharding. In the sharding-based approaches [20, 19, 26], the blockchain system is split into multiple smaller groups of peers, and each group operates in parallel on an independent version of the ledger. Despite its advantage of increasing the speed of the system on processing the transactions in parallel, existing sharding-based blockchains have $O(n)$ communication complexity for processing a single transaction, as well as the best case $O(\frac{b}{\log n})$ memory and time complexity at the Storage and View layers, respectively [26].

Proposed solution: In this paper, we propose *LightChain*, which is a permissionless blockchain defined over a Skip Graph-based peer-to-peer (P2P) Distributed Hash Table (DHT) overlay [32], with the goal of providing a consistent, communication and storage efficient blockchain architecture with fully decentralized and uniform block generation decision-making. *LightChain* is permissionless [33] as it allows every peer to freely join the blockchain system and be considered in the block generation decision-making, which is similar to the well-known blockchains like Bitcoin [1] and Ethereum [34]. At the Network layer, *LightChain* operates on top of a Skip Graph that is a DHT-based structured P2P system with a well-defined topology and deterministic lookup strategy for data objects. We model each peer, block, and transaction by a Skip Graph node. This idea enables participating peers to make their blocks and transactions addressable and efficiently accessible at the Network layer with the communication complexity of $O(\log n)$. In other words, each peer, block, and transaction is retrievable by exchanging at most $O(\log n)$ messages. Additionally, the latest state of each data object is retrievable with the same communication complexity and by querying the Skip Graph overlay directly. By the latest state of a data object, we mean the consideration of all the updates on that data object from the very first block to the most recent one on the ledger. For example, in the cryptocurrency applications where data objects are the peers' balance, the latest state corresponds to the most recent updated value of a peer's balance. This is in contrast to the existing solutions that require the peers to follow the ledger linearly, apply all the updates sequentially, and compute the latest state of a data object. As we elaborate in the rest of this paper, we utilize Skip

Graph due to its ability to represent each node with two independent identifiers. Nevertheless, *LightChain* can operate on any DHT with two independent identifiers.

To provide a time and bandwidth efficient consensus approach that is also fair, immutable, and secure, we propose Proof-of-Validation (PoV) as the Consensus layer strategy of *LightChain*. We say that a consensus approach is fair if each participating peer in the system has a uniform chance of being involved in the consensus regardless of its influence: e.g., processing power, available bandwidth, or stake. In addition, we say that a consensus approach is immutable if none of the (influential) peers in reaching a consensus can legitimately change the consensus at a later time after it is finalized. We say that a consensus approach is secure if the malicious peers are not able to generate and append an illegitimate transaction or block to the ledger. In PoV, the validation of each block is designated to a subset of the peers, which are chosen uniformly for each block based on its hash value (modeled as a random oracle), and are contacted efficiently using the structured Skip Graph overlay. Working in this fashion, *LightChain* enables improved decentralization of the block generation decision-making and deters the centralization monarchy. By the centralization monarchy, we mean the situation where the majority of block generation decision-makings are under the control of a small subset of special peers e.g., peers with a strong hashing power. *LightChain* preserves the integrity and consistency of the blockchain in the presence of colluding adversarial peers (e.g., Sybil adversary [35]) as well as selfish miners [29], as no peer can contribute to the decision making of two consecutive blocks generation.[1]We discuss these formally in the rest of this paper.

To improve the consistency of the ledger, *LightChain* governs a deterministic rule on resolving the forks at the Consensus layer. The main chain is always recognized in a deterministic fashion, and is followed by all the peers. Blocks on the other branches of a fork are discarded by the *LightChain* peers, i.e., block generation on those branches are rejected by the set of randomly assigned PoV validators, and hence by other peers of the system. This mechanism allows a block to be evaluated and finalized in a deterministic manner as the main chain's block once it is appended to the ledger and one other block comes after it, which is in contrast to the existing solutions that require appending several more subsequent blocks (e.g., around 6 blocks in Bitcoin [31]) to a new block for that block to be considered as a main chain's block.

To establish an efficient Storage layer policy, *LightChain* enables the peers to access the transactions and blocks in an on-demand basis using the efficient Skip Graph retrievability, rather than requiring them to store the entire ledger locally. Each peer is responsible for keeping a small subset of the randomly chosen blocks and transactions. This provides a storage load distribution among the participating peers. To provide better availability of blocks and transactions and tackle malicious peers, *LightChain* makes several copies of each block and

[1]In Section 5 we analyze this probability formally, and show that it happens only with a negligible probability in the system's security parameter.

transaction on different peers of the system, which is known as replication. Replication in *LightChain* is done in a way that it provides at least one copy of each block and transaction accessible at any time in expectation.

At the View layer, *LightChain* provides each new peer with a set of randomly chosen peers of the system that are named the *view introducers* of the new peer. The introducers of a new peer are drawn uniformly from the set of participating peers to share their view of the ledger with it. This is done to facilitate the bootstrapping of a new peer joining the system, and enable its immediate participation on the blockchain system without the need to verify the entire blockchain as opposed to the existing solutions. The randomized bootstrapping in *LightChain* takes $O(\log n)$ communication complexity and $O(n)$ time complexity. At the end of randomized bootstrapping, a peer obtains the updated view of the most recent state of all the participating peers in the system. However, as we stated earlier, obtaining a particular peer's state in *LightChain* takes the communication complexity of $O(\log n)$ and time complexity of $O(1)$, and without the need to track the ledger up to the most recent block. As presented later in this paper, *LightChain* determines the introducers in a way that the obtained view of a new peer towards the system is consistent with the view of the honest peers.

Contributions: The original contributions of this paper are as follows.

- To the best of our knowledge, this is the first study in the blockchain literature that improves the communication efficiency at the Network layer, the consistency and fairness at the Consensus layer, the memory efficiency at the Storage layer, and provides a more efficient bootstrapping at the View layer, altogether. With this aim, we propose *LightChain*, which a consistent, and communication and storage efficient permissionless blockchain with fully decentralized and uniform block generation decision-making that operates on top of a Skip Graph-based structured P2P overlay.

- *LightChain* is fair in the sense that each of the participating peers in the system has a uniform chance of being involved in the consensus regardless of its influence: e.g., processing power, available bandwidth, or stake.

- Having n peers and b blocks in the system, compared to the best existing solutions that require the storage and communication complexity of $O(\frac{b}{\log n})$ and $O(n)$ by maintaining many shards, respectively, our proposed *LightChain* requires $O(\frac{b}{n})$ storage on each peer, and incurs the communication complexity of $O(\log n)$ on generating a new block or transaction employing a new blockchain design approach.

- In our proposed *LightChain*, the transactions, blocks, as well as the latest state of the data objects are addressable within the network, and retrievable with the communication complexity of $O(\log n)$.

- We provide an analytical framework for the mathematical analysis of *LightChain*, showing how to set the operational parameters to achieve security, efficiency, and availability.

- We extended the Skip Graph simulator SkipSim [36] with the blockchain-based simulation scenarios, implemented and simulated the *LightChain*, and compared the experimental results with our proposed analytical framework. The analytical and experimental results are found to be consistent.

The related works are summarized in Section 2. We describe the preliminaries and our system model in Section 3. Our proposed *LightChain* is presented in Section 4. We describe the analytical and experimental results in Section 5, followed by conclusions in Section 6.

2 Related Works

In this section, we survey the existing blockchain solutions based on their contributions to each of the blockchain protocol stack's layers.

2.1 Network Layer

Dissemination of a new transaction or block in the existing blockchains is done via Broadcasting [1], Flooding [24], or Gossiping [25], which are epidemic disseminations with the communication complexity of $O(n)$ i.e., $O(n)$ message exchanges are required for a single block or transaction to be accessible by every peer of the system. On the other hand, our proposed *LightChain* applies a communication complexity of $O(\log n)$ messages to insert a new transaction or block in the Skip Graph overlay, and make it accessible by every peer of the system. Additionally, in our proposed *LightChain*, not only the blocks, but also the latest state of the data objects are addressable within the network, and retrievable with the communication complexity of $O(\log n)$. By the latest state, we mean the most recent appearance of that data object in a block on the ledger. By directly retrieving the latest state of a data object, in contrast to the existing blockchains, peers in *LightChain* are not required to keep searching and retrieving the most recent blocks frequently. Rather, they are able to search and retrieve the latest state of their data objects of interest on demand. For example, in cryptocurrency applications, a peer that is interested only in the latest balance state of another peer, performs a search within the Skip Graph overlay, and finds the latest balance state of the other peer within the blockchain.

2.2 Consensus Layer

Proof-of-Work (PoW): In PoW-based approaches [1, 37, 34, 11], the block generation is done by tweaking a parameter (i.e., nonce) of the block that makes the hash of it below a predefined difficulty level. Considering the hash values are drawn from a uniform distribution (i.e., random oracle model), reaching a hash value below the difficulty level requires a brute force approach over the input range [38]. The block generation decision-making in PoW is heavily correlated with the hash power, which sacrifices the fairness of the system in favor of the nodes with higher hash power [39]. Additionally, PoW is an inefficient

consensus solution due to its huge amount of energy consumption [40] e.g., the power requires to maintain the PoW on Bitcoin network is equal to the Ireland's electrical consumption [41].

Proof-of-Stake (PoS): In PoS-based approaches, the block generation decision-making is done by the stakeholders [12, 13, 14, 34, 42]. PoS approaches require a synchronized clock [43] among the peers, which applies an additional $O(n)$ communication complexity. PoS approaches also move the system towards the centralized monopoly of the peers with higher stakes and break down fairness and decentralization on block generation [12, 34, 42, 14, 13].

PoW-PoS Hybrid: To provide a balance between the computational inefficiency of PoW and communication inefficiency of PoS, hybrid PoW-PoS approaches are proposed. In PPCoin [15] the difficulty of PoW is adaptively determined for each peer based on its stake. Similar PoW-PoS hybrid approaches are proposed to combat the spammers in the email systems [44, 39, 45]. Proof-of-Activity (PoA) [16] is another hybrid approach where peers use PoW over empty blocks to determine the voting committee of the next block uniformly from the set of stake holders.

Compared to the existing PoW and PoS consensus solutions, our proposed Proof-of-Validation (PoV) is the only one that provides fairness, security, and immutability altogether. PoV is fair as it distributes the chance of participating in transactions' and blocks' validation decision-making uniformly among the participating peers, regardless of their influence in the system. In contrast to PoW-based consensus approaches, PoV is secure as malicious peers are not able to generate a validated transaction or block, even with large computational power. In contrast to the PoS-based approaches that are vulnerable to the posterior corruption attack, PoV is immutable and secure against such attacks. Posterior corruption attack happens when the majority of the committee members of an old block change their decision later on and create a (legitimate) fork from their generated block on the ledger [13]. This attack happens especially when the committee members of a block are coming out of stake, and hence do not have anything to lose [46]. In our proposed PoV, however, changing even one bit of a transaction or block changes its set of validators entirely. Hence, even the validators of a transaction or block are not able to change its content or fork another history later on.

Byzantine Fault Tolerance (BFT) Consensus and Sharding: In its classical form, the BFT-based consensus operates on voting nodes. Each node broadcasts its vote to the others, receives their votes, and follows the majority. BFT can tolerate up to $\frac{n}{3}$ of adversarial nodes [47]. Hyperledger [28, 27], Ripple [48], and Tendermint [49] support BFT-based consensus protocols. For example, in Ripple, during each epoch, each authority contacts a subset of other authorities as the trusted ones for voting. In sharding-based approaches the system is partitioned into disjoint subsets of peers, e.g., subsets of size $O(\log n)$ in Rapidchain [26]. Each subset is working on an independent version of the ledger using BFT in an epoch-based manner [19, 20]. Such epoch-based approaches and BFT apply an additional $O(n)$ communication overhead to the system. NEO [17] is another epoch-based blockchain that aims at resolving forks by Delegated

Byzantine Fault Tolerance (dBFT). In dBFT, the participating peers select a set of consensus peers and delegate the block generation decision making to them. Ontology [18] offers Verifiable Byzantine Fault Tolerance (VBFT), where the consensus peers of the next blocks are selected randomly from the set of stakeholders by applying a random function on the current block. Both dBFT and VBFT require the communication complexity of $O(n)$ for reaching consensus over a block. Snowflake is the consensus layer protocol of Avalanche [21], and acts similarly to our proposed PoV in the sense of randomly chosen peers for validation. However, in contrast to our proposed PoV that engages a small constant number of uniformly chosen peers, Snowflake requires the communal participation of all the online peers for reaching a consensus. Only a small set of trusted super-peers are participated in consensus protocol of BigChainDB [5]. BigChainDB establishes a variant of Paxos [50] consensus among the set of super-peers to elect the one that is responsible for writing to the ledger.

2.3 Storage Layer

Having b blocks in the system, existing blockchains like Bitcoin [1] and Ethereum [34], all require peers to keep an $O(b)$ storage. To moderate this linear storage complexity, Rollerchain [24] obligates peers to only hold a smaller subset of challenged blocks for generating the new ones. The subset, however follows a linear storage complexity in the number of blocks in the system i.e., $O(b)$, which is in contrast to our proposed *LightChain* that requires $O(\frac{b}{\log n})$ storage complexity on each peer. Additionally, Rollerchain lacks the storage load balancing as well as efficient block retrieval features, since blocks are not addressable within the network. Rollerchain also applies a noticeable communication overhead by including the exact copies of the challenged blocks into the newly generated block. Trustchain [25] aims to improve the storage load by making each peer to come with its own personal ledger. Each transaction is stored solely on its sender's and receiver's ledgers. In addition to the requirement of a globally synchronized clock and lack of replication, personal ledgers result in $O(n \times b)$ time complexity on generating new transactions. Similar personal ledger approach is also proposed in [51]. Personal ledgers also make the blockchain system not efficiently adaptable to the scenarios where a vast majority of the peers are working on a shared set of data, e.g., distributed database applications. By sharding the system into smaller groups of $O(\log n)$ peers that operate on disjoint ledgers, Rapidchain [26] requires each peer to keep $O(\frac{b}{\log n})$ blocks, but without an efficient retrieval feature. BigChainDB [5] provides a distributed database of super-peers (e.g., Cassandra [52]) that are the only ones responsible to keep the entire ledger. Ordinary peers can connect to the super-peers, read the ledger, and propose the transactions. Writing to the database (i.e., ledger) in BigChainDB, however, is only limited to a small set of super-peers that are assumed fully trusted. Compared to the existing solutions, our proposed *LightChain* requires $O(\frac{b}{n})$ storage complexity on each peer, and incurs the communication complexity of $O(\log n)$ on both generation and retrieval of the transactions and blocks, while it presumes a uniform chance for every par-

ticipating peer to be involved in the block generation decision-making.

2.4 View Layer

To the best of our knowledge, there is no existing secure and fast (i.e., $O(1)$ in time and $O(\log n)$ in communication) bootstrapping approach as we have in our proposed *LightChain*. Rather, almost all the existing blockchain architectures, including the Bitcoin [1] and Ethereum [34], require all the peers that are participating in the consensus protocol to construct their view locally by verifying the transactions on the ledger linearly. Having b blocks in the system, this local self-construction of view from scratch takes the time and communication complexity of $O(b)$. To improve the scalability of blockchain and boost up its transaction processing speed, side-chains are proposed as a view-layer solution [53], where a group of peers deviate from the main chain and create their own chain, proceed with their intra-group transactions for a while, and then close the side chain and summarize their turnover by submitting a few transactions into the main chain. Although the side-chains are running faster with significantly fewer peers than the main chain, they are prone to the efficiency problems of the main chain, such as forking. Likewise, as the side-chains grow in number, it is very likely for the sender and receiver of a transaction to reside on two different side-chains, which requires an inter-side-chain transaction. The inter-side-chain transactions should pass through the main chain with possibly two transactions [10] i.e., one deposit from sender's side-chain to the main chain, and one withdrawal from the main chain to the receiver's side-chain. This increases the number of the transactions by a factor of two and acts more of a hurdle for the main chain with advert scalability impact.

2.5 Blockchains in relation to the DHTs

Instead of storing blocks in a linked-list, Skipchain [22] provides a Skip List [54] representation of the blocks in the ledger. Skip List is the centralized analog of Skip Graph. Skipchain enables a peer to search for a specific block within its own local memory in the time complexity of $O(\log b)$, while it retains the communication complexity of $O(n)$ at the network layer. In the other words, in contrast to our proposed *LightChain* that enables peers to search for the blocks within the network in a fully decentralized manner, each peer of Skipchain is required to download the entire ledger on its own disk ($O(b)$ storage) and construct the Skip List locally to be able to search for a specific block in logarithmic time. Skipchain is a single-writer blockchain, i.e., only one entity is allowed to write on the blockchain entirely. This limiting assumption is used as appending a new block to Skipchain requires a deterministic knowledge of the meta-data of the immediately subsequent block. A blockchain-based decentralized access control management that does not require a trusted third party is proposed in [55]. The access granted by a user to a service is modeled as a transaction that is encrypted and stored on a DHT [56]. Users hold the pointers to the encrypted transactions on a blockchain and hence can revoke or change the access grants

Strategy	Network	Consensus	Storage	Clock-free
Bitcoin [1]	Broadcast	PoW	Full	Yes
BitCoin-NG [11]	Broadcast	PoW	Full	No
NEM [12]	Broadcast	PoS	Full	No
Snow White [13]	Broadcast	PoS	Full	Yes
Ouroboros [14]	Broadcast	PoS	Full	No
PPCoin [15]	Broadcast	PoW-PoS	Full	No
PoA-Bitcoin [16]	Broadcast	PoW-PoS	Full	Yes
NEO [17]	Broadcast	dBFT	Full	No
Ontology [18]	Broadcast	VBFT	Full	Yes
Elastico [19]	Broadcast	BFT	Full	No
Ripple [48]	Broadcast	BFT	Full	No
Tendermint [49]	Gossiping	BFT	Full	No
Hyperledger[27, 28]	Gossiping	BFT	Full	No
Omniledger [20]	Gossiping	BFT	Full	No
Avalanche [21]	Gossiping	Snowflake	Full	No
Skipchain [22]	Gossiping	BFT	Full	Yes
PeerCesus [23]	Flooding	PoW-PoS	Full	No
Rollerchain [24]	Flooding	PoW	Distributed	Yes
Trustchain [25]	Gossiping	PoW	Distributed	No
Rapidchain [26]	Gossiping	BFT	Distributed	No
BigChainDB [5]	Broadcast	Paxos	Distributed	No
LightChain	**DHT**	**PoV**	**Distributed**	**Yes**

Table 1: A comparison among a variety of the existing blockchain solutions. We assume that an approach supports distributed storage, if the storage load of blocks and transactions is distributed among all the participating peers in a policy-based manner, e.g., replication. Otherwise, we presume the full storage where peers collect and hold the blocks and transactions entirely on their local storage. We call a blockchain as clock-free, if it does not require the peers to be synchronized over a physical or logical clock.

later on. The blockchain in their proposed solution is mainly utilized as a service without the aim to improve its efficiency, and it is not constructed over the DHT.

Table 1 summarizes a variety of the existing blockchain solutions in comparison to our proposed *LightChain*.

3 Preliminaries and System Model

3.1 Skip Graph

Skip Graph [32] is a DHT-based distributed data structure that consists of nodes. A Skip Graph node is a standalone component with three attributes; a numerical ID, a name ID, and an (IP) address. The numerical and name IDs are known as the identifiers of the nodes. In order to join a Skip Graph, it is sufficient for each new node to know only one arbitrary node of the Skip Graph, which is called the introducer of that new node. As a result of the join protocol of the Skip Graph that is executed by the new node in a fully decentralized

manner, the new node obtains the attributes of $O(\log n)$ other nodes that are called the neighbors of the new node, where n is the number of Skip Graph nodes. Knowing its neighbors, each node is able to search and find the address of other nodes of Skip Graph that possess a specific numerical ID or name ID, by employing a search for numerical ID [32, 57], or a search for name ID [58] of those nodes, respectively. Both searches are done with the communication complexity of $O(\log n)$. As the result of the searches, if the targeted numerical ID or name ID of the search is available in the Skip Graph, the (IP) addresses of their corresponding nodes is returned to the search initiator. Otherwise, the (IP) addresses of the nodes with the most similar identifiers to the search target are returned.

3.2 Blockchain

A blockchain is a linked-list of blocks with immutable links from each block to its previous one [10, 59]. By immutable links, we mean that each block points back to the collision-resistant hash value of its previous block on the chain. The immutable links define an order of precedence over the chain of blocks, which implies that the transactions of a certain block are committed subsequent to the transactions of the previous blocks. Due to the immutable links, the blockchain is considered as an append-only database, and updating a block of the ledger by changing its content is not allowed, and considered as an adversarial act. An update on a block changes its hash value and makes the next subsequent block on the ledger not to pointing to this block's hash value anymore, which corresponds to a disconnection on the ledger. To re-establish the connectivity between the updated block and its subsequent block, the pointer on the subsequent block needs to be refreshed with the new hash value of the updated block. This, in turn, changes the hash value of the subsequent block and breaks the ledger from a new point onward (i.e., the subsequent block). Hence, re-establishing the connectivity after an update on a single block requires refreshing the hash pointers on all the subsequent blocks. In the existing blockchains, re-establishing the connections after an update on a block is correlated with a success probability, e.g., solving a computationally hard problem [1] or getting the consent of a specific subset of peers [42]. This correlation makes re-establishing the connectivity of ledger upon changing the content of even a single block a computationally hard problem due to the collision-resistance of the hash functions.

3.3 Notations

In this paper, we call the last block that is appended to the blockchain as the *current tail* of the blockchain, which is also the tail of the ledger. The first block of a blockchain is known as the *Genesis* block, which is also the head of the linked-list of the ledger. We also define the *previous* relationship as the immutable links from each block to its previous block on the ledger. Blockchain defines a partial ordering of the blocks on the ledger based on the previous relationship. We say that block $blk1$ is the *immediate predecessor* of

the $blk2$, if $blk2$ points back to the hash value of $blk1$ as its previous block on the ledger. In this situation, $blk2$ is the *immediate successor* of $blk1$. In this paper, we consider that a block is *committed* to the blockchain if it is being written by the consensus layer protocol of the blockchain to its storage, i.e., the block passes the defined consensus verification and is being appended to the tail of the ledger. We denote the system's security parameter and the system's identifier size by λ and s, respectively. Also in this paper, we denote the hash function $H : \{0,1\}^* \rightarrow \{0,1\}^s$ as a random oracle.

3.4 System Model

In our system model, each peer corresponds to a device connected to the Internet (e.g., a laptop, smartphone, smart TV) that executes an instance of the *LightChain* protocol. As detailed in Section 4, a Skip Graph overlay of peers is constructed by representing each peer as a Skip Graph node. We assume that each participating peer joins the Skip Graph overlay using the Skip Graph join protocol in a fully decentralized manner and by knowing one peer of the system [32]. Both identifiers (i.e., name ID and numerical ID) of peers are the hash value of their public key using a collision-resistant hash function. Following this convention, in this paper, we refer to a peer by its identifier, which corresponds to its name/numerical ID. We consider the system under churn [60], i.e., the participating peers are dynamic between offline and online states. We assume the existence of a churn stabilization strategy [61, 62] that preserves the connectivity of the Skip Graph overlay under churn. We denote the *System Capacity* by n, and define it as the maximum number of registered peers in the system, i.e., $n = O(2^s)$. We consider all the participating peers as probabilistic Turing machines that run in a polynomial time in the security parameter of the system i.e., their running time is $O(\lambda^c)$ for some constant $c > 0$. We make this assumption essentially for the reason that participating peers should be able to execute $O(n)$ cost protocols. Following this assumption, n is a polynomial in λ, which results in $s << \lambda$. Similarly, we denote the *Block Capacity* by b, and define it as the maximum number of the generated blocks in the system. Similarly, we also consider b as a polynomial in λ, which results in $\frac{b}{n}$ to be a polynomial in the security parameter of the system.

In our system model, we assume that each peer is participating in the blockchain by a set of *assets* as well as a *balance*. The assets set corresponds to the data that the peer initially registers on the blockchain via a transaction, and is able to update it later on by generating new transactions. The balance of a peer is used to cover its transaction generation fees. Although the assets and balance are the same in cryptocurrency applications, nevertheless, we consider them as two distinct attributes of each participating peer in general form, considering other potential applications such as distributed databases. We consider a transaction as a state transition of the assets of the transaction's owner. View of a participating peer in our system model towards the blockchain is a table of $(numID, lastblk, state, balance)$ tuples. Each tuple represents the view of the peer with respect to another peer of the system with the numerical ID of

numID. The *lastblk* represents the hash value of the last committed block to the blockchain that contains the most recent transaction of that peer. The view of the associated peer with respect to the current state of the assets of another peer and its remaining balance are represented by *state* and *balance*, respectively. By the current state, we mean the most recent values of the assets of the peer considering all the generated transactions by that peer from the Genesis block up to the current tail of the blockchain.

3.5 Adversarial Model

We define the availability of the blockchain as the blocks being accessible in a timely fashion [63]. We define the integrity of the blockchain as the property that views of the peers towards the blockchain are not being changed, except by appending a new block to the current tail of the blockchain solely by the peers that are included in the consensus protocol. We assume the existence of a Sybil adversarial party [35] that adaptively takes control over a fraction f of peers in the system. We define the honest peers as the ones that follow the *LightChain* protocol, and the adversarial peers as the ones that deliberately deviate from the *LightChain* protocol collectively at arbitrary points. Adversarial peers aim to jointly attack the availability and integrity of the system.

3.6 Authenticated Search

We assume that the search queries over the Skip Graph overlay are authenticated by an authentication mechanism in the presence of the described adversarial peers [64]. By the authenticated searches, we mean that the validity of the search results is publicly verifiable through a search proof that is generated by the signing keys of the participating peers on the search path. The search proof also contains the attributes of the peers on the search query path (e.g., identifier and (IP) address) with the last node on the search path considered as the search result.

4 LightChain: A Permissionless Blockchain over Skip Graph

4.1 Overview

The *LightChain* protocol is executed independently by each participating peer. In *LightChain*, we employ a Skip Graph DHT overlay to establish a blockchain. The peers, as well as the transactions and blocks, are indexed as Skip Graph nodes. Each peer invokes the insertion algorithm of Skip Graph [32] using its own identifiers and (IP) address and joins the system. Both identifiers of a peer (i.e., its name ID and numerical ID) are hash value of its public key (i.e., verification key). As a result of joining the Skip Graph overlay, each peer knows logarithmically other peers, which enables it to efficiently search for any

other peer of the system with the communication complexity of $O(\log n)$. Upon joining the Skip Graph overlay, the peer creates its view of the blockchain using *LightChain*'s randomized bootstrapping feature without the need to download and process the entire ledger.

In *LightChain*, a transaction represents a state transition of the assets of a peer, which is denoted by the *owner* peer of that transaction. For example, in the cryptocurrency applications, the asset of a peer is its monetary wealth, and a transaction models a monetary remittance, which represents the state transition of the monetary wealth of the owner affected by the remittance. The owner peer casts the state transition into a transaction, computes the identifiers of validators, searches for the validators over Skip Graph overlay, and asks them to validate its transaction. In order to be validated, each transaction needs to be signed by a system-wide constant number of validators, where their identifiers are chosen randomly for each transaction to ensure security. In addition to security, the idea of validating transactions makes participating nodes in the block generation needless of going through the validation of individual transactions.

Once the transaction gets validated, the owner inserts it as a node into the Skip Graph overlay, which makes it searchable and accessible by any other peer. The insertion of the transaction is done by invoking the insertion protocol of Skip Graph using the transaction's identifiers but the (IP) address of the owner peer itself. As we explain later, the identifiers of a transaction are related to its hash value. The Skip Graph peers route the messages on behalf of the transactions they hold. This idea is similar to the other existing DHTs like Chord [65] and Pastry [66]. This feature enables *LightChain* peers to search and find the new transactions. Upon finding new validated transactions, each peer is able to cast them into blocks, go through the validation procedure (similar to the transactions' case), and insert the validated block into the Skip Graph overlay. Each transaction's owner then removes its transaction node from the overlay once it is successfully included in a validated block (for the sake of efficiency). The idea of representing each transaction and block by a Skip Graph node results in any search for the peer or the transactions and blocks that it holds to be routed to the peer's (IP) address, rendering them accessible by every other peer in a fully decentralized manner. Hence, in *LightChain*'s Skip Graph overlay, there exist three types of nodes: peers, transactions, and blocks. In other words, the Skip Graph overlay acts as a distributed database of the transactions and blocks that are owned by peers, which enables each peer to efficiently search for any transaction or block with the communication complexity of $O(\log n)$. The *previous* relationship of blocks stored in a distributed manner on distinct peers defines a blockchain. By making the blocks and transactions efficiently retrievable by search, the participating peers are not required to keep or download the entire ledger. In *LightChain*, each block or transaction is replicated by its owner and validators to support availability, accessibility, and fault tolerance. By means of searchable blocks and transactions as well as replication, in *LightChain* we introduce the idea of distributed storage layer for the blockchain where participating peers in the consensus only need to keep and maintain a subset of the blocks, and not the ledger entirely. In the rest of this section, unless stated

View Layer:	Randomized-Bootstrapping Time Complexity: O(n) Communication Complexity: O($\log n$)
Storage Layer:	Randomized-Replication Storage Complexity: O($\frac{b}{n}$)
Consensus Layer:	Proof-of-Validation (PoV) Time Complexity: O(1) Communication Complexity: O($\log n$)
Network Layer:	Using Skip Graph DHT Communication Complexity: O($\log n$)

Figure 1: *LightChain*'s protocol stack and its contributions to each layer of the blockchain architecture. The reported asymptotic complexities for the Network layer are per transaction or block, and for other layers are per node.

otherwise, by the term node, we mean a peer.

As an incentive mechanism, *LightChain* employs a monetary balance for each participating peer to exchange with other peers and cover the operational fees of appending data to the blockchain [1]. *LightChain* rewards the peers' contribution on maintaining the connectivity of the system, providing validation service, and generating blocks. Moreover, *LightChain* encourages honest peers to audit other peers, by rewarding the detection and report of adversarial acts. Malicious behavior is penalized by *LightChain* upon detection, and the adversarial peers are blacklisted and gradually isolated from the system. Figure 1 summarizes the *LightChain*'s contributions to each layer of the blockchain architecture.

4.2 Structure of Transactions and Blocks

A *LightChain* transaction, tx, is represented by a ($prev$, $owner$, $cont$, $search\_proof$, h, σ) tuple, where $prev$ is the hash value of a committed block to the blockchain. We use the $prev$ pointer for each transaction tx to define an order of precedence between tx and all the blocks and transactions in the blockchain without the need of any synchronized clock. The block that is referred by $prev$ takes precedence over tx. All the transactions included in the $prev$ block are assumed to be committed before tx in the essence of time. Following the same convention, all the blocks and transactions that precede $prev$, also precede tx. The $owner$ represents the identifier of the owner node in the Skip Graph overlay that generates the transaction tx. Equating the name ID and numerical ID of the peers with the hash value of their public key, $owner$ refers to either of the name ID or numerical ID of the owner peer. The $cont$ field of a transaction denotes the state transition of the assets of the owner node. The contribution is a general term that covers a vast variety of the blockchain applications that *LightChain* is applicable on. For example, in cryptocurrency applications, the state of peers corresponds to their wealth, and a transaction represents a monetary remittance between two peers. In such applications, $cont$ includes the remittance value as well as the identifier of the receiver peer, to whom the transaction owner aims to

transfer the fund. The *search_proof* field of a transaction is the authenticated proof of searches over the peers of the Skip Graph overlay to find the validators of the transaction tx, as explained before. The h field of the transaction tx is the hash value of the transaction, which is computed as shown by Equation 1. The σ field of the transaction tx contains the signatures of both the owner as well the validators on its hash value h. The owner's signature is for the sake of authenticity, and to prevent adversarial peers from masquerading as honest peers and submitting a transaction on behalf of them. The validators' signature is a part of *LightChain*'s consensus strategy, and is explained within our proposed Proof-of-Validation consensus approach.

$$h = H(prev||owner||cont||search\_proof) \tag{1}$$

A *LightChain* block *blk* is defined by a ($prev$, $owner$, $\mathcal{S}$, $search\_proof$, h, σ) tuple, which is similar to the transaction structure of *LightChain* except that $\mathcal{S}$ represents the set of all the transactions that are included in the block *blk*. The h field of block *blk* is its hash value, which is computed as shown by Equation 2. The σ field contains the signatures of both the block's owner as well as the block's validators on its hash value (i.e., h).

$$h = H(prev||owner||\mathcal{S}||search\_proof) \tag{2}$$

4.3 Network Layer: Skip Graph overlay of peers, transactions, and blocks

In our proposed *LightChain*, we represent each peer, transaction, and block by a Skip Graph node. This way, all the peers, transactions, and blocks are addressable within the network. In other words, participating nodes (i.e., peers) in *LightChain* exploit the Skip Graph overlay to search for each other, as well as each others' blocks and transactions. Both the numerical ID and name ID of the peers are the hash value of their public key using a collision-resistant hash function. As in a Skip Graph, nodes' identifiers define the connectivity; hence, considering the hash function as a random oracle results in the uniform placement of peers in Skip Graph overlay, which limits the adversarial power on tweaking the Skip Graph topology for its advantage.

The numerical ID of a transaction or a block in the Skip Graph overlay is its hash value (i.e., h). The name ID of a transaction or a block is its corresponding *prev* field value. This regulation enables peers to traverse the *LightChain*'s ledger in both forward and backward directions. Following this convention, in *LightChain*, having a block with numerical ID (i.e., the hash value) of h and previous pointer value of *prev*, the (IP) address of the peers that hold the immediate predecessor block are obtained by performing a search for numerical ID of *prev* in the Skip Graph overlay. Similarly, the (IP) address of the peers holding the immediate successor transaction(s) or block(s) in the blockchain are obtainable by performing a search for name ID of h over the Skip Graph overlay. This follows the fact that all the immediate successors of a block have h as their

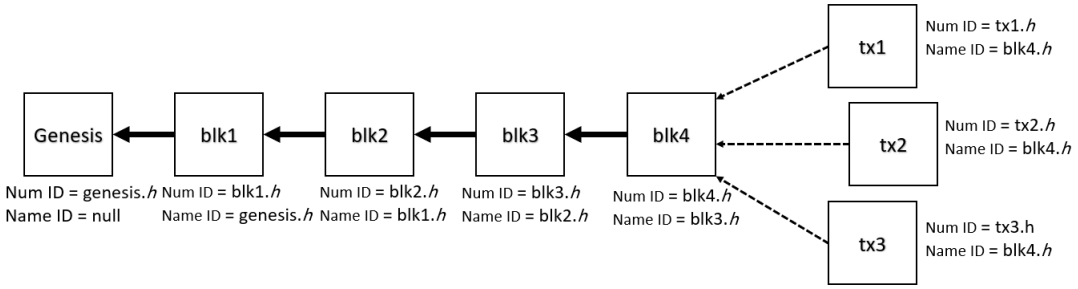

Figure 2: The *LightChain* regulation on name IDs and numerical IDs. Numerical ID (i.e., Num ID) of a block or transaction is its hash value, and name ID is its corresponding *prev* value.

name ID. This feature of the *LightChain* enables the peers to efficiently update their view towards the tail of the blockchain by performing a search for the name ID of their local tail. The search returns all the blocks that are appended subsequently to their local tail, as well as all the new validated transactions that are waiting to be included in blocks. Additionally, using this feature, a peer does not need to store the entire blockchain locally. Rather, having only a single block of the ledger enables the peer to efficiently retrieve the predecessor and successor blocks to it with the communication complexity of $O(\log n)$.

Figure 2 illustrates this convention of *LightChain*, where a peer that only has $blk2$ is able to efficiently retrieve its immediate predecessor (i.e., $blk1$) by searching for the numerical ID [32] of its *prev* value (i.e., $blk1.h = blk2.prev$) in a fully decentralized manner. The search is responded by the owner [2]of $blk1$ with its (IP) address, and hence the predecessor of $blk2$ (i.e., $blk1$) is retrievable efficiently by directly contacting its owner. Similarly, the peer that only possesses $blk2$ is able to perform a search for name ID [58] over its hash value (i.e., $blk2.h$) to retrieve the immediate successor block that comes after $blk2$. As the result of the search for name ID of $blk2.h$, owner of $blk3$ responds to the search initiator peer with its (IP) address, and $blk3$ is retrievable efficiently by directly contacting its owner. In the case where a single block has several successor blocks, the search initiator receives a response from each of the immediate successor block's owners. In the example of Figure 2, considering $blk4$ as the current tail of the blockchain, as discussed later in this section, the newly generated transactions that succeed $blk4$ (i.e., $tx1$, $tx2$, and $tx3$) are efficiently retrievable by performing a search for the name ID using $blk4.h$.

4.4 Consensus Layer: Proof-of-Validation (PoV), fair, efficient, immutable, and secure consensus

Proof-of-Validation (PoV) is our proposed consensus approach of *LightChain*, and is employed to validate the generated transactions and blocks. Once a

[2]Considering the replication of blocks, the search is responded by the either the owner, or one of the replicas. We introduce the replication of blocks and transactions later in this section.

transaction or block is validated by PoV, it is considered legitimate by all the participating peers. PoV is fair as each participating peer in the system has a uniform chance of being involved in the consensus regardless of its influence. PoV is efficient as it requires only $O(\log n)$ communication complexity for validating a single transaction or block. PoV is immutable as none of the influential peers in reaching a consensus can legitimately change the consensus at a later time after it is finalized. Finally, PoV is secure as malicious peers are not able to commit an invalid transaction or block to the blockchain. We analyze the security and immutability of PoV in Section 5. A transaction or block is considered as validated once it successfully passes the PoV consensus. Note that a validated transaction's contribution is not considered effective and authoritative unless it is included in a validated block that is committed to the blockchain. To validate each transaction or block, PoV provides a set of randomly chosen validators for the sake of evaluation as detailed in the followings.

4.4.1 Transaction Generation and Validation

PoV considers a transaction as valid if its hash value h is signed by t (randomly chosen) validators, where t is a constant protocol parameter, which is called the *Signatures Threshold*. For a transaction tx, the numerical ID of each validator is chosen uniformly as shown by Equation 3, where v_i is the numerical ID of the i^{th} validator in the Skip Graph overlay. In order to provide security, efficiency, and availability for the system, *LightChain* only allows a transaction's owner to iterate i over $[1, \alpha]$, where α is another constant protocol parameter, which is called the *Validators Threshold*. We formally discuss this in Section 5, and develop a formulation for deciding on the proper values of the *Signatures Threshold* and *Validators Threshold* considering the security, efficiency, and availability of system.

$$v_i = H(tx.prev||tx.owner||tx.cont||i) \tag{3}$$

The transaction's owner then conducts a search for numerical ID of the validator (i.e., v_i) within the Skip Graph overlay. If there exists a peer with the numerical ID of v_i in the overlay, the owner receives its (IP) address. Otherwise, it receives the (IP) address of the peer with the largest available numerical ID that is less than v_i. Both cases are supported with an authenticated search proof that is generated by the Skip Graph peers on the search path, and is delivered to the owner. The authenticated proof of the search for numerical ID of the i^{th} validator is denoted by $search\_proof_i$, which also contains all the (IP) addresses and identifiers of the Skip Graph peers on the search path. The last peer on the search path of v_i is designated as the i^{th} validator. The transaction's owner then adds the authenticated search proof for all the validators to the transaction, computes its hash value h as specified by Equation 1, signs the hash value, and appends her signature to σ. The transaction's owner then contacts the validator asking for the validation of the tx. During the validation, the validators evaluate the soundness, correctness, and authenticity of tx, as well as the balance compliance of the its owner to cover the fees. As the validation

result for tx, the transaction owner either receives a signature over h or $\perp$ from each contacted validator.

Soundness: A transaction tx is said to be sound if it does not precede the latest transaction of the transaction's owner on the blockchain. By not preceding the latest transaction of the same owner, we mean its *prev* should point to the hash value of a validated and committed block on the ledger with no transaction of the transaction's owner in any of the subsequent blocks. In other words, soundness requires the newly generated tx transaction to succeed all of the previously registered transactions of its owner on the blockchain. This is both to counter double-spending from the same set of assets, as well as to make the validation of a transaction a one-time operation, i.e., the owner of a validated tx transaction is able to append it to the blockchain as long as it does not generate any new transaction on the blockchain that precedes tx based on *prev*. Considering the soundness, at most one of the concurrently generated and validated transactions of a peer has the chance to be included into a new block. As once one of its transactions is included in a block, the others become unsound, cannot be included in the same block or further blocks, and should go over re-validation. Therefore, in addition to prevent double spending, soundness provides a uniform chance for the transaction generators to include their transaction into each new block. We elaborate more on this criteria when we discuss block validation.

Correctness: For a transaction tx to be correct, its contribution field (i.e., *cont*) should represent a valid state transition of the owner's assets. The compliance metric is application dependent. For example, in cryptocurrency applications, for a transaction to be correct, the owner's account should have enough balance to cover the remittance fee (i.e., the contribution).

Authenticity: The evaluation of authenticity is done by checking the correctness of h based on Equation 1, verifying σ for the inclusion of a valid signature of the transaction's owner over h, and verifying $search\_proof$ for all the validators of tx. The validator rejects the validation of tx as unauthenticated if any of these conditions is not satisfied.

Balance Compliance: As an incentive mechanism to participate in the validation, *LightChain* considers a validation fee in the favor of the t validators of the transaction tx that sign its hash value and make it validated. Also, *LightChain* considers a routing fee in the favor of all the Skip Graph peers that participate in finding the transaction's validators, i.e., the peers that their identifiers are included on the search path of every validator according to the $search\_proof$, excluding the validator and the owner itself. A transaction tx passes the balance compliance part of validation if its owner has enough balance to cover the validation and routing fees. The balance compliance validation is done based on the view of the validator towards the blockchain. Both the routing and validation fees are fixed-value protocol parameters, and are the incentive mechanism for the peers to perform the routing and validation honestly [1, 11, 12]. The fees also prevent Sybil adversarial peers on indefinitely generating transactions by circulating the adversarial balance among themselves and continuously congesting the system with the validation of adversarial transac-

tions.

If tx is sound, correct, authenticated, and its owner has a balance compliance to cover the fees, the validator signs h, and sends the signature to its owner, who then includes the validator's signature inside σ. For a transaction tx to be considered as validated, PoV requires the owner to include t valid signatures issued by the validators in the $search\_proof$. The validated tx transaction is inserted as a Skip Graph node by its owner, which makes it accessible by other participating peers of the system to be included in a block. The numerical ID of tx is h, and the name ID of tx is $prev$, which enables any Skip Graph peer to conduct a search for name ID on the hash value of any ledger's block within the Skip Graph overlay and find all the new transactions that are pointing back to that block.

4.4.2 Block Generation and Validation

We call a peer that generates blocks, a block owner. Once a block owner collects at least min_tx newly generated transactions that have not been included into any validated block that has been committed to the blockchain, it casts them into a new block blk, and sends the block for validation. By casting transactions into blk we mean including the collected transactions into $\mathcal{S}$ set as discussed in Equation 2. min_tx is an application-dependent fixed-value parameter of *LightChain* denoting the minimum number of the transactions that should be included in a block. In contrast to the transaction owners that have more flexibility on choosing their transaction's $prev$ pointer, the block owners should always set the $prev$ pointer of their block to the current tail of the blockchain. Similar to the transactions, in PoV we say a block blk is validated if its hash value (i.e., h) is being signed by t randomly chosen PoV validators. To have blk validated, the block owner computes the numerical ID of the i^{th} validator as shown by Equation 4.

$$v_i = H(prev||owner||\mathcal{S}||i) \tag{4}$$

Similar to the transaction validation, the block owner searches for validators in the Skip Graph overlay, and completes the structure of blk by including the search proof for validators into $search\_proof$, computing the block's hash value (i.e., h), and including its own signature over h into σ. The block owner then contacts each of the validators and asks for the validation. Consistent with the transaction validation, a block owner is only allowed to iterate over Equation 4 for $i \in [1, \alpha]$. As the validation result for blk, the block owner either receives a signature over h or $\perp$ from each contacted validator. If the block owner receives t signatures over h from its PoV validators, it is said that the block passed the PoV validation. On receiving a validation request for a block blk, each of its PoV validators checks the authenticity and consistency of blk itself, as well as the authenticity and soundness of all transactions included in $\mathcal{S}$ (as discussed earlier). The authenticity evaluation of blocks is done similar to the transactions.

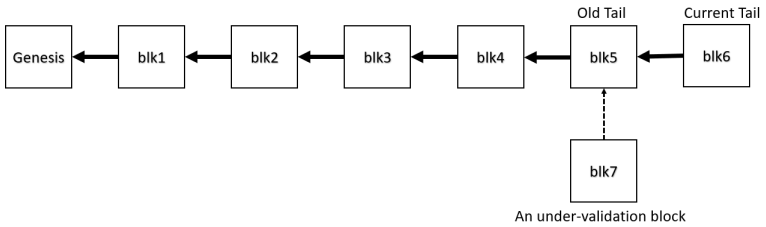

Figure 3: An example of a potential fork. Validation of block $blk7$ is rejected and terminated by its validators at any state of the validation upon detection of the new block, $blk6$, as the new tail of the blockchain.

Consistency: A block blk is said to be consistent, if its *prev* pointer points to the current tail of the blockchain; otherwise it is inconsistent. By the current tail of the blockchain, we mean the most recent view of the validators towards the tail of the chain. The inconsistencies among validators' views are handled by our proposed *fork-free mechanism* later. However, it is likely for the current tail of the blockchain to be updated during the validation of a newly generated block. Although randomly chosen PoV validators of a block evaluate its consistency during the validation phase, nevertheless, the update on the current tail of the blockchain makes the block inconsistent during the validation procedure. Validating such an inconsistent block emerges a fork on the blockchain. To tackle this problem, once any of the randomly chosen PoV validators detects a potential fork at any step of the validation, it terminates the validation with a rejection, informing the owner. By a potential fork, we mean the situation where another block outpaces an under-validation block and becomes the new tail of the blockchain. This implies that the validators of a block need to keep their view of the blockchain's tail updated by continuously performing a search for the name ID of the hash value of the current tail (during the validation process only), which returns all the blocks and transactions that immediately succeed the tail. In this manner, upon any update on the current tail, the new tail is returned as the result of the search. A potential fork example is illustrated by Figure 3 where $blk7$ is undergoing the validation but its validation is terminated with rejection as soon as any of its randomly chosen PoV validators detects that another block (i.e., $blk6$) has outpaced $blk7$ in validation and became the new tail block of the blockchain.

If the block's structure is authenticated, consistent, and all the transactions in $\mathcal{S}$ are sound and authenticated, the validator signs h, and sends the signature to the block's owner, who includes the validator's signature inside σ. PoV considers a block blk as validated if σ field contains t valid signatures on h value. After the blk gets validated, its owner inserts it into the Skip Graph overlay as a node. As the incentive mechanism of *LightChain*, owner of a block receives a block generation reward once its block gets validated and committed to the blockchain. The block generation reward is a fixed-value parameter of *LightChain* that acts both as an incentive mechanism for encouraging the peers to participate progressively in generating blocks, as well as a mean for wealth

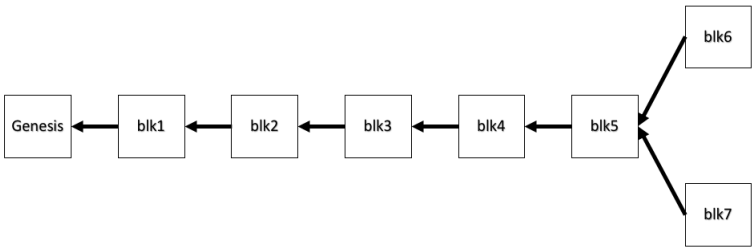

Figure 4: Using our *fork-free* mechanism, whichever of the simultaneously validated $blk6$ or $blk7$ has the lowest hash value wins the fork, is followed by every participating peer. The knocked-out block owners remove their blocks from the Skip Graph overlay, update their transactions set, and restart the validation procedure.

creation. In this paper, we assume that the generation reward for a block is larger than its validation and routing fees. This is done to enable peers to participate in the block generation immediately after they join the system.

Fork-free mechanism: To resolve the forks caused by the simultaneously validated blocks, *LightChain* governs a fork-free mechanism, which is a deterministic approach that instructs all the peers to solely follow the block with the lowest hash value upon a fork. For example, in the snapshot of Figure 4 in the fork that is caused by simultaneous validations of the blocks $blk6$ and $blk7$ by disjoint set of PoV validators, whichever of $blk6$ or $blk7$ that has the lowest hash value is presumed as the one committed to the blockchain, and is followed by all the peers of the system. Upon a fork, we call the block with the lowest hash value as the winner block, and the other participating blocks of the fork as the knocked-out ones. The knocked-out block owners remove their block from the Skip Graph overlay, update their set of transactions by dropping the transactions that are included in the winner block, adding the new transactions to reach the min_tx threshold, and restart the validation procedure. The knocked-out block owners neither gain any block generation reward nor lose any balance because of the fees, as these fees and rewards are not effective unless the block is successfully committed to the blockchain, i.e., the block passes the PoV validation, wins the possible forks, and is appended to the current tail of the blockchain. In order to ensure that a newly appended validated block blk to the ledger does not undergo any further fork rivalry, and is considered committed, effective, and finalized, *LightChain* waits for only one further block to be appended subsequently to blk. In this way, all the forks at the depth of the blk are considered as potential forks, and are rejected by the consistency checking mechanism of PoV. Once a block blk is committed to the blockchain, the contributions and fees of transactions in $\mathcal{S}$, as well as the fees and rewards associated with blk itself become effective.

4.5 Storage Layer: Replication for better efficiency and availability

In *LightChain*, each transaction or block is stored in the local storage of its associated owner, and presented as a Skip Graph node, which makes it efficiently searchable by all the participating peers in the system. Hence, peers do not need to store or download the entire ledger. Rather, they access the transactions and blocks in an on-demand manner, i.e., a peer searches for a transaction or block upon a need and retrieves it efficiently from the overlay. For further efficiency, a transaction owner should remove its transaction from the overlay once it is included in a committed block, to be discarded from the list of transactions that are waiting to be placed into the blocks. We assume the peers in *LightChain* are subject to churn, i.e., volatile between online and offline states [60]. To provide availability of the transactions and blocks under churn, all the randomly chosen PoV validators of a transaction or block also act as its corresponding replicas by storing a copy of it in their local storage, representing it as a node in the overlay, and being responsive to the other peers' queries over it. In Section 5, we show that the *Signatures Threshold* parameter of *LightChain* (i.e., t) is chosen in a way that it results in at least one available replica for each transaction and block under churn, in expectation.

4.6 View Layer: Randomized Bootstrapping, trusted, consistent, and efficient view synchronization

View of a peer in *LightChain* is a table of ($numID$, $lastblk$, $state$, $balance$) tuples. Each view table entry represents a single peer of the system with the numerical ID of $numID$, the current state of assets that is determined by the $state$, and the remaining balance of $balance$. The $lastblk$ represents the hash value of the last block on the blockchain that contains the most recent transaction of that peer. We define the *view introducers* of a new peer as the set of randomly chosen peers that share their view of the blockchain with the newly joined peer. Upon joining the overlay, a new peer computes the numerical IDs of its view introducers based on Equation 5, where $new\_peer.numID$ is the numerical ID of the new peer and $view\_intro_i$ is the numerical ID of the i^{th} view introducer of it. We employ the hash function as a random oracle to obtain uniformly random view introducers' numerical IDs.

$$view\_intro_i = H(new\_peer.numID||i) \tag{5}$$

The new peer then conducts a search for numerical ID of $view\_intro_i$ within the overlay, contacts the peer in the search result, and obtains its view of the blockchain. The new peer continues in this manner by iterating over i until it obtains t consistent views. As we show later, we determine t and α in such a way that a new peer obtains t consistent views of the honest peers by iterating i over $[1, \alpha]$.

4.7 Direct retrieval of the latest state

Each transaction that appears on a committed block to the ledger contains the latest update on the transaction owner's state of assets. *LightChain*'s approach on representing each block by a Skip Graph node makes the blocks addressable, searchable, and efficiently retrievable within the network. Tracking the updates on the entire view of other peers' assets, hence, requires a peer to keep its local view updated with the new blocks, which is a plausible assumption in the majority of the existing solutions. However, in addition to sequentially seeking the new blocks and updating view accordingly, *LightChain* enables each peer to directly retrieve the latest assets' state of another peer of interest without the need to keep track of the new blocks on the ledger. This is done by the additional representation of each block with multiple Skip Graph nodes i.e., one per each transaction included in the block. As each of these additional Skip Graph nodes represents one of the transactions of the same block, we call them the associated *transaction pointers* of that block. In this approach, each transaction tx that is included into a committed block blk is represented by a transaction pointer node (i.e., *pointer*). The name ID and numerical ID of the transaction pointer node are set as $pointer.nameID = tx.owner$ and $pointer.numID = blk.h$, respectively. Setting the numerical ID of a transaction pointer to its associated block's hash value is for the sake of security, and to provide a tie between each pointer and the block it points to. The transaction pointer nodes associated with each block are inserted by the block's owner and replicated on the block's PoV validators. In this manner, a peer that is solely interested in knowing the latest state of another peer's assets, for example, $tx.owner$, performs a search for a transaction pointer with the name ID of $tx.owner$ as the search target within the Skip Graph overlay. The search is answered by either the owner of blk or one of its PoV validators that all keep a copy of blk (i.e., the block that contains the latest update on the assets' state of $tx.owner$). To keep track of the latest updates over the assets, both the owner and validators of a block should take down each of its associated transaction pointers once an update on the corresponding assets appears on a newer committed block to the ledger. Taking down a pointer node from the overlay is simply done by performing the Skip Graph node deletion operation [32] by the owner and each of the validators in a fully decentralized manner. This is for the sake of better efficiency of the search, and to make sure that the transaction pointers always point to the most recent states. Not dropping the pointers after a new update is counted as misbehavior, which we address it by the misbehavior detection strategy of *LightChain*. To address the network asynchrony, however, the block owner and PoV validators are allowed to take down the pointers within at most a certain number committed blocks after a new transaction on the associated set of assets happens. This allows them to have enough time to discover the new updates without being subject to misbehavior. The length of the block interval (i.e., number of blocks between two transaction pointers over the same set of assets) is a constant protocol parameter that is application dependent.

4.8 Motivating honest behavior and misbehavior detection:

The block generation reward and the routing and validation fees constitute the incentive mechanism of the *LightChain* for the peers to retain their honest behavior i.e., following the *LightChain*'s architecture and protocol as described in this section. In this paper, we assume that the block generation reward is greater than the routing and validation fees. We establish this assumption to motivate any peer to retain honest behavior from the time it joins the system by enabling it to participate in block generation and gain the block generation reward. The counterpart of honest behavior is the *misbehavior*, which we define it as any sort of deviation from the described *LightChain*'s protocol and architecture. As detailed earlier, for the transactions and blocks that are gone through the consensus layer, we consider the randomly chosen PoV validators to check the submitted transaction or block against the misbehavior. As we analyze in Section 5, we choose the system's security parameter (i.e., λ) as well as PoV operational parameters (i.e., α and t) in a way that an adversarial peer cannot convince the PoV validators on a misbehavior unless with a probability of at most $2^{-\lambda}$. In addition to the countermeasures established by PoV, we also introduce *misbehavior detection* as an extra level of adversarial countermeasure, especially for the adversarial actions that are not gone through the PoV e.g., direct submission of an invalid block to the ledger. In our proposed misbehavior detection, each peer of *LightChain* acts as an auditor for other peers' behavior and gains a *misbehavior audition reward* by reporting their misbehavior. As an auditor, any peer should be able to evaluate a block or transaction in the same way that its PoV validators do during the validation. Any deviation from *LightChain*'s protocol that fails the auditor's evaluation is considered as misbehavior, e.g., the invalid signature of validators, lack of t signatures on the hash value, and validating an invalid block or transaction. We specified the first two cases earlier in this section. The last case (i.e., validating an invalid block or transaction) happens when an adversarial transaction or block owner finds t randomly chosen malicious PoV validators who deviate from the validation protocol and sign an invalid block or transaction, e.g., a double-spending transaction. Although as we stated earlier, we determine PoV operational parameters in a way that such an attack cannot happen unless with a probability of at most $2^{-\lambda}$, nevertheless, the misbehavior detection of *LightChain* provides an extra level of security to ensure that even if such an attack happens, the invalid transaction or block does not persist on the ledger.

Upon a misbehavior detection, the auditor generates a transaction with the evidence of the misbehavior in the contribution field. The transaction then goes through the same PoV validation process as described earlier, except that the validators verify the correctness of the transaction as the correctness of the reported evidence. Once the transaction is validated and placed into a committed block to the blockchain, the guilty peer is penalized by the misbehavior penalty fee, routing fee, and validation fee that it is made to pay to the owner (i.e., auditor), routers, and validators of the transaction, respectively. Misbehavior

fee is another constant parameter of *LightChain* that is application dependent. Once misbehavior is recorded for a peer on a committed block, its identifier is blacklisted. The blacklisted peers are isolated by honest peers i.e., any incoming message from the blacklisted peers is discarded by honest peers. This eventually results in the blacklisted peers being excluded from the overlay, which causes the blacklisted peers to never being selected as a validator as they no longer are a part of the overlay from the honest peers' point of view. A blacklisted peer appearing in an authenticated search proof implies a malicious router peer on the search path that is caught and blacklisted accordingly.

The detailed pseudo-code descriptions of *LightChain*'s algorithms are presented in the Appendix.

5 Analytical and Performance Results

5.1 Formal Mathematical Analysis

In this section, we analyze the necessary mathematical conditions on picking the operational system parameters: the *Signatures Threshold* t and the *Validators Threshold* α. There exist three main considerations: (1) With respect to the security, an adversary that controls f fraction of all the peers should be able to fork a history or validate adversarial blocks or transactions with the probability of at most $2^{-\lambda}$, where λ is the system's security parameter. (2) With respect to the availability, *LightChain* should provide the availability of at least one replica of each block and transaction, at any given time in expectation. (3) With respect to the efficiency, the honest peers who follow the *LightChain* protocol find t honest validators within at most α trials for validating their transactions and blocks. We aim at choosing the minimum possible values of t and α that satisfies the mentioned security, availability, and efficiency constraints, since this means a lower communication overhead imposed to the system.

In our analysis, we assume the worst case such that there is no churn for the adversarial peers and all the adversarial peers are under the control of the same adversarial party. For simplicity, we assume a uniform failure model for the honest peers of the system, such that at any point in time the probability of failure of an honest peer is denoted by q, which is independent of the others. Based on these assumptions, the expected number of online peers at any time is denoted by n_o as determined based on the Equation 6. As is represented later in this section, we implement more realistic churn models in our simulations and show that the results match our analysis.

$$n_o = n(f + (1 - f)(1 - q)) \tag{6}$$

Security Analysis: Given a certain value of α, we indicate t_m as the minimum value of t that yields in a probability of less than or equal to $2^{-\lambda}$ for the adversary to find t adversarial validators within at most α trials. For example, assuming that $\lambda \geq 40$, it yields in the adversarial success probability of at most 2^{-40}. Let V_m be a random variable that denotes the number of adversarial

validators within at most α trials. We aim at achieving $Pr(V_m \geq t_m) \leq 2^{-\lambda}$. Equation 7 represents the evaluation of the cumulative distribution function of V_m at t_m. For large values of n, Equation 7 is approximated by Equation 8, which denotes a Hypergeometric distribution [67], where ψ is the standard normal distribution function [68]. Equation 9 represents a lower bound on the value of t_m that results in the $Pr(V_m < t_m) \geq 1 - 2^{-\lambda}$. The lower bound obtained from this equation is independent of the system capacity (i.e., n) as well as the churn of the peers (i.e., q).

$$Pr(V_m < t_m) = \frac{\sum_{i=0}^{t_m-1} \binom{nf}{i} \binom{n(1-f)(1-q)}{\alpha-i}}{\binom{n_o}{\alpha}} \tag{7}$$

$$\approx \psi(\frac{t_m - \alpha f - 1}{\sqrt{\alpha f(1-f)}}) \tag{8}$$

$$t_m \geq (\sqrt{\alpha f(1-f)} \times \psi^{-1}(1 - 2^{-\lambda})) + \alpha f + 1 \tag{9}$$

For a certain value of α, choosing $t \geq t_m$ results in a less than $2^{-\lambda}$ probability for an adversarial peer to find t adversarial validators within at most α trials, and validate an adversarial transaction or block. We also obtain a lower bound value for α based on Equation 9. The constraint of $t_m \leq t \leq \alpha$ implies a lower bound on α as shown by Equation 10.

$$\alpha \geq \frac{(\sqrt{f} + \sqrt{f \times (\psi^{-1}(1 - 2^{-\lambda}))^2 + 4}))^2}{4(1-f)} \tag{10}$$

Efficiency Aspect: We determine the value of t in such a way that finding t *honest* validators is efficiently achievable within at most α trials in expectation. As shown by Equation 11, we denote by p the Bernoulli probability of choosing an honest validator for a given transaction or block uniformly at random. The uniform distribution of p follows from the random oracle model.

$$p = \frac{n \times (1-f) \times (1-q)}{n_o} \tag{11}$$

Given a certain value of α, we indicate the minimum number of randomly chosen honest validators within α trials by t_h. Let X_i be a random variable denoting the number of trials on the validators' identifiers until the i^{th} honest validator is reached. X_i follows a geometric distribution with the success probability of p, and the expected value of $\frac{1}{p}$. As shown by Equation 12, we aim to choose t_h in such a way that it enables an honest peer to find t_h honest validators within at most α trials in expectation. Since X_i values are independent and identically distributed random variables, Equation 12 is simplified to Equation 13, which results in an upper bound value of $t_h \leq p \times \alpha$ as shown by Equation 14. For a certain value of α, one should choose $t \leq t_h$ to find t honest validators in

expectation.

$$E[\sum_{i=1}^{t_h} X_i] \leq \alpha \tag{12}$$

$$\sum_{i=1}^{t_h} E[X_i] = \sum_{i=1}^{t_h} \frac{1}{p} \leq \alpha \tag{13}$$

$$t_h \leq p \times \alpha \tag{14}$$

Availability Aspect: The t PoV validators and the owner of a block or transaction are also designated as its replicas. To provide data availability for *LightChain* under churn, we choose t in such a way that at least one replica is available for each transaction or block in a timely fashion in expectation. Let t_a be the minimum number of replicas that provides the availability of at least one replica under the uniform failure probability of q. It is trivial that $t_a \leq t+1$ as we should not replicate any more rather than on the owner as well as the t PoV validators of a transaction or block. The replication policy of *LightChain* on the owner as well as the randomly chosen PoV validators each with the failure Bernoulli probability of q corresponds to a Binomial distribution of the replicas availability. Having t_a replicas (including the owner), the expected number of available replicas at any given time is derived from the expected value of the Binomial distribution of t_a trials with the success probability of $1-q$, which is equal to $\frac{t_a}{1-q}$. Equation 15 represents the availability constraint of *LightChain*, which results in the lower bound of t_a. Excluding the owner from t_a, and replicating a block or transaction on its t_a-1 randomly chosen validators results in the expected availability of at least one copy in a timely fashion. Note that in cases where the minimum expected availability of k replicas is desirable (i.e., $k > 1$), Equation 15 is easily adoptable by replacing 1 in the nominator with k.

$$t_a \geq \frac{1}{1-q} \tag{15}$$

Putting all together: Having our security parameter (i.e., λ) and the fraction of adversarial peers (i.e., f) determined, we obtain a lower bound value for α form Equation 10 that satisfies our security constraint. In order to satisfy the security requirement we need $t \geq t_m$, for efficiency we need $t \leq t_h$, and for availability we need $t \geq t_a$. Putting all these aspects together results in a permissible range for t that is shown by Equation 16.

$$max\{t_a - 1, t_m\} \leq t \leq t_h \tag{16}$$

5.2 Experimental Evaluation

Setup: To simulate and evaluate *LightChain*, we extended the Skip Graph simulator SkipSim [36] by enhancing it with three types of nodes on the overlay: peers, transactions, and blocks. In SkipSim, the time is divided into the fixed discrete intervals of one hour. To simulate the arrivals and departures of the

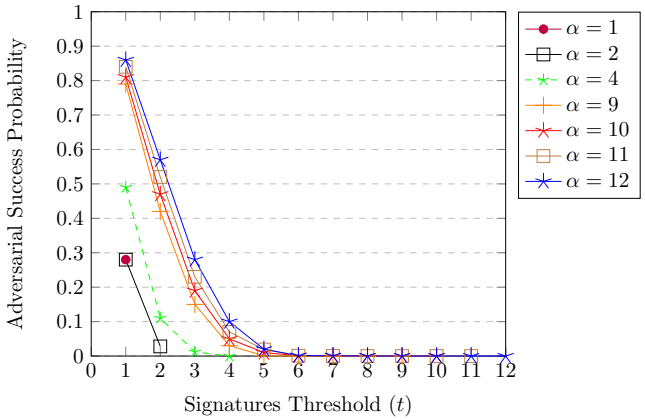

Figure 5: The security of *LightChain* with respect to the colluding adversarial peers. X-axis shows the Signatures Threshold parameter of *LightChain*, i.e., t. Y-axis shows the adversarial peers' success probability on finding t adversarial PoV validators.

peers to and from the system (i.e., churn) in a realistic manner, we follow the extracted Weibull-based churn model [60] over a long-term study of the P2P systems. In this churn model, a peer shows the periodic states of online and offline with the expected duration of 2.7 and 10 hours, respectively. The overlays are initially empty, and peers join the overlay with an average of about 1000 peers per hour. Consistent with the reported statistics of the Bitcoin that is provided in [69, 70], in our *LightChain* implementation on SkipSim, while a peer is online it generates a single transaction per hour. Also in our simulations, we considered 16.5% of the total registered peers as colluding adversarial peers (i.e., $f = 0.165$), which corresponds to the largest fraction of colluding hash power in the Bitcoin network [13]. We simulated *LightChain* for 100 randomly generated Skip Graph overlay topologies, each with the system capacity of $n = 10,000$ peers. By randomly generated overlays we mean the randomized set of peers' identifiers as well as the overlay's connectivity. Each topology was simulated for 48 hours (i.e., time slots). Likewise, for the sake of simulation, we set min_tx $= 1$. Therefore, each topology generates an average of $100K$ blocks during the simulation time.

Security Aspect: Figure 5 shows the success probability of the colluding adversarial peers for different values of α and $t \leq \alpha$. We consider the adversarial success as forking a history or validating an adversarial block or transaction by finding t adversarial PoV validators within at most α trials. As shown by Figure 5, for each value of α, with the growth of t from 1 to α, the success probability of adversarial peers drops exponentially and converges to zero. Applying the simulation parameters on Equation 10 results in an estimated lower bound of $\alpha \geq 9.61$ based on our analytical framework. Following Equation 9 from the analytical framework, α values of 10, 11, and 12, result in the t_m lower bounds of 9.05, 9.59, and 9.89, respectively, which is supported by Figure 5. For $\alpha \geq 10$, the adversarial success probability drops essentially to zero as their associated t value goes beyond their corresponding obtained t_m from the framework, e.g.,

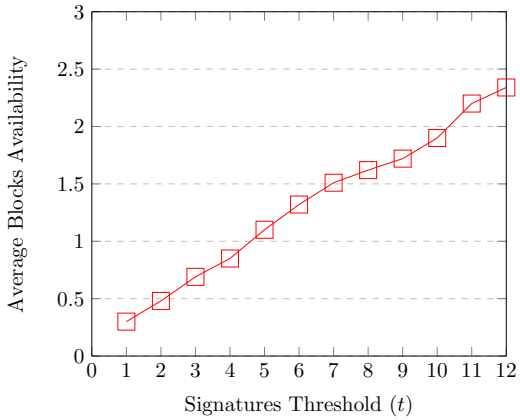

Figure 6: Availability of *LightChain*. X-axis represents the Signatures Threshold parameter of *LightChain*, i.e., t. Y-axis represents the average number of available replicated copies of each block at each time slot.

$t \geq 9.89$ for $\alpha = 12$ results in the adversarial success probability of essentially zero.

Availability Aspect: Figure 6 illustrates the average availability of the blocks as t grows. By the blocks' availability, we mean the average number of available replicas of each block in the system at each time slot. The average is taken over 48 hours of the simulation time. The t parameter that corresponds to the number of PoV validators for each block of *LightChain* also represents the number of replicated copies of that block. As shown by Figure 6, the average availability of the blocks increases linearly with respect to t. This linear behavior is supported by the Binomial distribution of the randomized replication policy of *LightChain* over the validators. Modeling the churn by a uniform model with the average online and offline periods of about 2.7 and 10 hours, the uniform failure probability of each peer is $q = 0.78$. Considering q as a Bernoulli probability, the expected number of available replicas is obtained as $q \times t$, which is linear in t. Following Equation 15 from our analytical framework, $t_a \geq 4.55$. As illustrated in Figure 6, for $t \geq t_a$, the average availability of one block over the entire simulation time is obtained. Taking $t = 10$ following the security discussion above provides average block availability of about 2.

Efficiency Aspect: Figure 7 represents the efficiency of *LightChain* with respect to the honest peers who follow the protocol on finding t honest PoV validators. A point (x, y) on this figure is interpreted as *x randomly chosen honest PoV validators are obtainable on the average within y trials*. We obtain an expected bound of $t_h \leq 0.84 \times \alpha$ from Equation 14 of our analytical framework considering the described simulated churn. Concerning honest peers to efficiently find t honest validators within at most α trials, for a certain value of α, t_h gives an expected upper bound on the value of t. For example, for $\alpha = 12$, t_h gives an expected upper bound of 10.08, which is consistent with the simulation results on Figure 7.

Analytical versus Simulation Results: As discussed earlier, considering

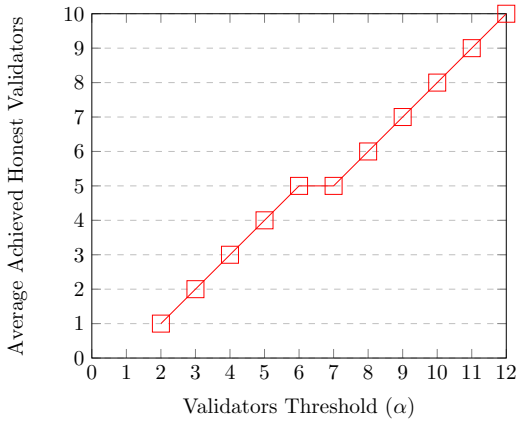

Figure 7: Efficiency of *LightChain* with respect to the honest peers. X-axis represents the Validators Threshold parameter of *LightChain*, i.e., α. Y-axis shows the average number of honest PoV validators that are obtainable with α trials.

our simulation setup and assuming $\lambda = 48$ as our security parameter, we obtain $\alpha \geq 9.61$ from our analytical framework. Selecting $\alpha = 12$, results in $t_m \geq 9.89$, $t_a \geq 4.55$, and $t_h \leq 10$. Following Equation 16 from our analytical framework, we obtain $t = 10$ as the proper Signatures Threshold value for the Validators Threshold value of $\alpha = 12$ that satisfies the security, availability, and efficiency constraints of *LightChain*, which is also supported by our simulation results in Figures 5, 6, and 7, respectively.

5.3 Asymptotic Analysis

Having the system capacity of n peers, a new peer joins the Skip Graph overlay using the original join protocol of the Skip Graph [32], which has the communication complexity of $O(\log n)$. Generating a new transaction or block takes the maximum of α searches over the overlay for the sake of PoV validation, as well as an insertion in the overlay once it got validated. This results in the overall communication complexity of $O((\alpha + 1) \log n)$ for generating a single transaction or block, which is simplified to $O(\log n)$ considering α as a system-wide constant parameter. To reduce the number of searches, the participating peers on the search for numerical ID path can be alternatively contacted as the PoV validators, which is supported by the randomized identifier assignment of the peers. In this manner, $O(\frac{\log n}{\alpha})$ searches for validators are required. Although it does not change the communication complexity asymptotically, it reduces the number of searches. For example, based on our simulation results, in a Skip Graph of 10K peers for $\alpha = 12$ and $t = 10$, an average of one search is needed to obtain 10 honest validators. PoV validation of a single transaction or block is done via sending a validation result message, which takes the communication complexity of $O(1)$. Also, both generation and validation of a new transaction or block take constant computational operations that are a function of the *LightChain*'s operational parameters (e.g., verifying t signatures from

PoV validators), and takes $O(1)$ asymptotic time complexity. For the sake of *LightChain*'s Randomized-Bootstrapping, a peer needs to search for at most α view introducers within the overlay, which takes the communication complexity of $O(\log n)$. **Thus we conclude that the communication complexity of *LightChain* is $O(\log n)$ per operation.** Similarly, having b blocks in the system, following the replication on validators policy of *LightChain*, **the expected storage complexity of each peer is $O(\frac{b}{n})$.** This follows the fact that the validators of each block are chosen uniformly from a random oracle modeled hash function. Based on the simulation results, with about 100K generated blocks during the 48 hours of simulation, the average standard deviation of the number of replicated blocks that each peer holds is about 0.033, which corresponds to a uniform load distribution of blocks over the peers.

6 Conclusion

To improve the communication and storage efficiency, and solve the convergence to centralization and consistency problems of the existing blockchain solutions, in this paper, we proposed *LightChain*, which is a novel blockchain architecture that operates over a Skip Graph-based P2P DHT overlay. In contrast to the existing blockchains that operate on epidemic data dissemination, *LightChain* provides addressable peers, blocks, and transactions within the network, which makes them efficiently accessible in an on-demand manner. Using *LightChain*, no peer is required to store the entire ledger. Rather each peer replicates a random subset of the blocks and transactions and answers other peer's queries on those. *LightChain* is a fair blockchain as it considers a uniform chance for all the participating peers to be involved in the consensus protocol regardless of their influence in the system (e.g., hashing power or stake). To improve the consistency of the blockchain, *LightChain* governs a deterministic fork-resolving policy.

We analyzed *LightChain* both mathematically and experimentally regarding its operational parameters to achieve the security, efficiency, and availability. Having n peers and b blocks in the system, compared to the existing blockchains that require the storage and communication complexity of $O(b)$ and $O(n)$, respectively, *LightChain* requires $O(\frac{b}{n})$ storage on each peer, and incurs the communication complexity of $O(\log n)$ on generating a new transaction and block. As future work, we plan to extend our analysis to include game-theoretical aspects of *LightChain* (e.g., rewards and fees).

Acknowledgement

The authors thank Ali Utkan Şahin, Yüşa Ömer Altıntop, and Ece Tavona for their contributions to the SkipSim implementation.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[2] A. Reyna, C. Martín, J. Chen, E. Soler, and M. Díaz, "On blockchain and its integration with iot. challenges and opportunities," *Future Generation Computer Systems*, 2018.

[3] M. A. Khan and K. Salah, "Iot security: Review, blockchain solutions, and open challenges," *Future Generation Computer Systems*, 2018.

[4] Z. Ma, M. Jiang, H. Gao, and Z. Wang, "Blockchain for digital rights management," *Future Generation Computer Systems*, 2018.

[5] T. McConaghy, R. Marques, A. Müller, D. De Jonghe, T. McConaghy, G. McMullen, R. Henderson, S. Bellemare, and A. Granzotto, "Bigchaindb: a scalable blockchain database," *white paper, BigChainDB*, 2016.

[6] P. Jiang, F. Guo, K. Liang, J. Lai, and Q. Wen, "Searchain: Blockchain-based private keyword search in decentralized storage," *Future Generation Computer Systems*, 2017.

[7] S. Delgado-Segura, C. Pérez-Sola, G. Navarro-Arribas, and J. Herrera-Joancomartí, "A fair protocol for data trading based on bitcoin transactions," *Future Generation Computer Systems*, 2017.

[8] K. Leng, Y. Bi, L. Jing, H.-C. Fu, and I. Van Nieuwenhuyse, "Research on agricultural supply chain system with double chain architecture based on blockchain technology," *Future Generation Computer Systems*, 2018.

[9] H. A. Kalodner, M. Carlsten, P. Ellenbogen, J. Bonneau, and A. Narayanan, "An empirical study of namecoin and lessons for decentralized namespace design." in *WEIS*, 2015.

[10] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer *et al.*, "On scaling decentralized blockchains," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016.

[11] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "Bitcoin-ng: A scalable blockchain protocol." in *NSDI*, 2016.

[12] J. M. BloodyRookie, Gimre, "Nem technical reference," *https://nem.io/wp-content/themes/nem/files/NEM˙techRef.pdf*, 2018.

[13] P. Daian, R. Pass, and E. Shi, "Snow white: Provably secure proofs of stake," Cryptology ePrint Archive, 2016, https://eprint.iacr.org/2016/919.

[14] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual International Cryptology Conference*. Springer, 2017.

[15] S. King and S. Nadal, "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake," *https://peercoin.net/assets/paper/peercoin-paper.pdf*, 2012.

[16] I. Bentov, C. Lee, A. Mizrahi, and M. Rosenfeld, "proof of activity: Extending bitcoin's proof of work via proof of stake [extended abstract] y," *ACM SIGMETRICS Performance Evaluation Review*, 2014.

[17] NEO, "Neo whitepaper," [Online; accessed 19-December-2018]. [Online]. Available: http://docs.neo.org/en-us/whitepaper.html

[18] ONT, "Ontology whitepaper," [Online; accessed 23-December-2018]. [Online]. Available: https://dev-docs.ont.io

[19] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *SIGSAC*. ACM, 2016.

[20] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *IEEE SP*, 2018.

[21] T. Rocket, "Snowflake to avalanche: A novel metastable consensus protocol family for cryptocurrencies," 2018.

[22] K. Nikitin, E. Kokoris-Kogias, P. Jovanovic, N. Gailly, L. Gasser, I. Khoffi, J. Cappos, and B. Ford, "Chainiac: Proactive software-update transparency via collectively signed skipchains and verified builds," in *USENIX Security 17*, 2017.

[23] C. Decker, J. Seidel, and R. Wattenhofer, "Bitcoin meets strong consistency," in *ICDCN*. ACM, 2016.

[24] A. Chepurnoy, M. Larangeira, and A. Ojiganov, "A prunable blockchain consensus protocol based on non-interactive proofs of past states retrievability." *arXiv preprint arXiv:1603.07926*, 2016.

[25] P. Otte, M. de Vos, and J. Pouwelse, "Trustchain: A sybil-resistant scalable blockchain," *Future Generation Computer Systems*, 2017.

[26] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *CCS*. ACM, 2018.

[27] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *EuroSys*. ACM, 2018.

[28] C. Cachin, "Architecture of the hyperledger blockchain fabric," in *Workshop on distributed cryptocurrencies and consensus ledgers*, vol. 310, 2016.

[29] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," *Communications of the ACM*, 2018.

[30] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *Big-Data congress*. IEEE, 2017.

[31] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey," *International Journal of Web and Grid Services*, 2018.

[32] J. Aspnes and G. Shah, "Skip graphs," *ACM TALG*, 2007.

[33] K. Wüst and A. Gervais, "Do you need a blockchain?" *IACR Cryptology ePrint Archive*, 2017.

[34] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, 2014.

[35] J. R. Douceur, "The sybil attack," in *International workshop on peer-to-peer systems*. Springer, 2002.

[36] Y. Hassanzadeh-Nazarabadi, A. Küpçü, and Ö. Özkasap, "Skipsim: An offline and scalable skip graph simulator," https://github.com/yhassanzadeh13/SkipSim, 2018.

[37] M. Jakobsson and A. Juels, "Proofs of work and bread pudding protocols," in *Secure Information Networks*. Springer, 1999.

[38] D. Kraft, "Difficulty control for blockchain-based consensus systems," *Peer-to-Peer Networking and Applications*, 2016.

[39] B. Laurie and R. Clayton, "Proof-of-work proves not to work; version 0.2," in *Workshop on Economics and Information, Security*, 2004.

[40] M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. bft replication," in *International workshop on open problems in network security*. Springer, 2015.

[41] K. O'Dwyer and D. Malone, "Bitcoin mining and its energy footprint," in *IET*. The Institution of Engineering & Technology, 2014.

[42] V. Buterin and V. Griffith, "Casper the friendly finality gadget," *arXiv preprint arXiv:1710.09437*, 2017.

[43] L. Lamport, "The implementation of reliable distributed multiprocess systems," *Computer Network*, 1978.

[44] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *Annual International Cryptology Conference*. Springer, 1992.

[45] D. Liu and L. J. Camp, "Proof of work can work," *WEIS*, 2006.

[46] A. Poelstra *et al.*, "Distributed consensus from proof of stake is impossible," *URL: https://download. wpsoftware. net/bitcoin/old-pos. pdf*, 2014.

[47] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *JACM*, 1980.

[48] D. Schwartz, N. Youngs, A. Britto *et al.*, "The ripple protocol consensus algorithm," *Ripple Labs Inc White Paper*, 2014.

[49] J. Kwon, "Tendermint: Consensus without mining," *Retrieved May*, vol. 18, p. 2017, 2014.

[50] L. Lamport *et al.*, "The part-time parliament," *ACM TOCS*, 1998.

[51] E. Harris-Braun, N. Luck, and A. Brock, "Holochain-scalable agentcentric distributed computing," *Alpha*, 2018.

[52] A. Cassandra, "Apache cassandra," *Website. Available online at http://planetcassandra. org/what-is-apache-cassandra*, 2014.

[53] A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timón, and P. Wuille, "Enabling blockchain innovations with pegged sidechains," *http://www.opensciencereview.com/papers/123/enablingblockchain-innovations-with-pegged-sidechains*, 2014.

[54] W. Pugh, "Skip lists: a probabilistic alternative to balanced trees," *Communications of the ACM*, 1990.

[55] G. Zyskind, O. Nathan *et al.*, "Decentralizing privacy: Using blockchain to protect personal data," in *Security and Privacy Workshops*. IEEE, 2015.

[56] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *International Workshop on Peer-to-Peer Systems*. Springer, 2002.

[57] Y. Hassanzadeh-Nazarabadi, A. Küpçü, and Ö. Özkasap, "Locality aware skip graph," in *IEEE ICDCSW, 2015*.

[58] ——, "Laras: Locality aware replication algorithm for the skip graph," in *IEEE NOMS*, 2016.

[59] M. Etemad and A. Küpçü, "Efficient key authentication service for secure end-to-end communications," in *ProvSec*. Springer, 2015.

[60] D. Stutzbach and R. Rejaie, "Understanding churn in peer-to-peer networks," in *SIGCOMM*. ACM, 2006.

[61] R. Jacob, A. Richa, C. Scheideler, S. Schmid, and H. Täubig, "Skip+: A self-stabilizing skip graph," *JACM*, 2014.

[62] Y. Hassanzadeh-Nazarabadi, A. Küpçü, and Ö. Özkasap, "Interlaced: Fully decentralized churn stabilization for skip graph-based dhts," *arXiv preprint arXiv:1903.07289*, 2019.

[63] M. T. Goodrich and R. Tamassia, *Introduction to computer security*. Pearson, 2011.

[64] K. Needels and M. Kwon, "Secure routing in peer-to-peer distributed hash tables," in *Symposium on Applied Computing*. ACM, 2009.

[65] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM Computer Communication Review*, 2001.

[66] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*. Springer, 2001.

[67] W. L. Harkness, "Properties of the extended hypergeometric distribution," *The Annals of Mathematical Statistics*, 1965.

[68] D. P. Bertsekas and J. N. Tsitsiklis, *Introduction to probability*. Athena Scientific Belmont, MA, 2002.

[69] "Global bitcoin nodes distribution," https://bitnodes.earn.com/, accessed: 24-09-2018.

[70] "Blockchain charts," https://www.blockchain.com/en/charts, accessed: 24-09-2018.

A LightChain's algorithms details

In this appendix, we represent the detailed descriptions of the *LightChain*'s algorithms in a bottom-up manner, i.e., we first show the basic low-level algorithms that act as the building blocks of the high-level ones, and then move to the explanation of the high-level algorithms that operate on the top of those building blocks. The Verify algorithm (Algorithm 1) verifies the authenticity of the input search proof, and generates a misbehavior transaction upon receiving an unauthenticated search proof (see Section 4.8 for more details). The TXB-Generation algorithm (Algorithm 2) is called whenever a peer aims to generate a transaction or block, and it uses the Verify algorithm as a sub-routine. Algorithms 3, 4, and 5 are sub-routines that evaluate the soundness, correctness, and authenticity of a transaction as described in Section 4.4, respectively. Algorithm 5 also evaluates the authenticity of blocks, which is done in an identical manner as for the transactions.

Using these building blocks, the Audit algorithm (Algorithm 6) evaluates the validity of a given transaction or block based on the view of the peer that invokes it, and generates a misbehavior transaction on receiving an invalid transaction or block (see Section 4.8 for further details). The Audit algorithm is used as a sub-routine in the ViewUpdate algorithm (Algorithm 7), which performs randomized bootstrapping for a new peer that joins the system (see Section 4.6) as well as updating the view of the existing peers in the system. For an already joined peer to the system, a single call to ViewUpdate updates the view of the peer towards the tail of the blockchain by one block, e.g., a peer with a view that is three blocks behind the current tail of the blockchain needs to invoke ViewUpdate three times to reach the current tail of the ledger. *LightChain* peers invoke ViewUpdate periodically to update their view towards the blockchain. The frequency of ViewUpdate execution is application dependent. ViewUpdate audits the newly generated transactions and blocks against the misbehavior, and generates a new block by invoking the TXB-Generation algorithm upon collecting min_tx newly generated transactions. Algorithm 8 evaluates the balance compliance of a transaction owner to cover the routing and validation fees based on the view of the PoV validator that invokes it. Finally, Algorithm 9 is executed by a PoV validator peer and represents the PoV validation procedure of a transaction or block.

Algorithm 1: Verify

Input: proof of search $search\_proof$, numerical ID of the peer $numID$, routing table of the peer $table$, local view of the peer on blockchain's tail c_tail

Output: boolean $result$

// Validate the search proof based on the validation mechanism of underlying authenticated search proof

if *search_proof is authenticated* **then**
    $result = true$;
else
    $result = false$;
    // Misbehavior detection (see Section 4.8)
    find the guilty node and report it in $cont$;
    // Generate a misbehavior transaction
    TXB-Generation($numID$, $table$, c_tail, $cont$);

Algorithm 2: TXB-Generation

Input: identifier of the owner $owner$, routing table of the owner $table$, previous pointer $prev$, contributions $cont$ **or** transaction set $\mathcal{S}$
Output: a new transaction **or** block $txblk$

```
while i ∈ [1, α] do
    // Search for the i^th validator within overlay
    v_i = H(prev||owner||cont/S||i);
    search_proof_i = searchForNumericalID(v_i, table);
    if Verify(search_proof_i, owner, table, prev) then
        add search_proof_i to search_proof;
    increase i;
include prev and search_proof into txblk;
include cont/S into txblk;
compute the hash and include it in txblk.h;
sign txblk.h;
include signature in txblk.σ;
send txblk for validation to the validators;
obtain the validation signatures from validators;
if t validators signatures on txblk obtained then
    include validators signature in txblk.σ;
    insert txblk into overlay as a node;
    if txblk is a block then
        if txblk is knocked-out in a fork then
            drop txblk from the overlay;
            terminate;
        else
            // Adding transaction pointers for direct state retrieval (See
               Section 4.7 for more details)
            for tx ∈ txblk.S do
                insert a transaction pointer to tx.owner into the overlay;
    if txblk is a transaction then
        while txblk is not included in a block do
            wait;
        drop txblk from the overlay;
```

Algorithm 3: isSound

Input: transaction tx, view of the auditor/validator $view$
Output: boolean $result$
// Retrieve the address of the last block of that contains the most recent transaction of the tx's owner
1 ($lastblk$, $state$, $balance$) = $view.get(tx.owner)$;
2 **if** *$tx.prev$ points to a predecessor of $lastblk$* **then**
  // tx is not sound as it violates the total ordering among the transactions of $tx.owner$
3 $result = false$;
4 **else**
5 $result = true$;

Algorithm 4: isCorrect

Input: transaction tx, view of the auditor/validator $view$
Output: boolean $result$
// Retrieve the state of the transaction owner from the view of the auditor/validator peer
1 ($lastblk$, $state$, $balance$) = $view.get(tx.owner)$;
2 **if** *$tx.cont$ contains a valid transition of $state$* **then**
3 $result = true$;
4 **else**
5 $result = false$;

Algorithm 5: isAuthenticated

Input: transaction/block $txblk$, numerical ID of the peer $numID$, routing table of the peer $table$, local view of the peer on blockchain's tail c_tail
Output: boolean $result$
// Result is initially true and is set to false upon detection of an authenticity violation
$result = true$;
if $txblk.h \neq H(prev||owner||cont/\mathcal{S}||search\_proof)$ **then**
    // Invalid hash value
    $result = false$;
else if *valid signature of* $txblk.owner$ *on* $txblk.h \notin txblk.\sigma$ **then**
    // Missing a valid signature of the owner
    $result = false$;
else
    // Check the number of PoV validators that signed the transaction/block
    $validatorCounter = 0$;
    for $i \in [1, \alpha]$ **do**
        $v_i = H(prev||owner||cont/\mathcal{S}||i)$;
        if $search\_proof_i \in search\_proof \wedge Verify(search\_proof_i,\ numID,$ $table, c\_tail) \wedge$ *signature of* i^{th} *validator on* $txblk.h \in txblk.\sigma$ **then**
            $validatorCounter++$;
            if $validatorCounter == t$ **then**
                break;
    if $validatorCounter < t$ **then**
        $result = false$

Algorithm 6: Audit

Input: transaction/block $txblk$, the auditor view $view$, numerical ID of the auditor $numID$, routing table of the auditor $table$, local view of the auditor on blockchain's tail c_tail

Output: boolean $result$

```
1  if txblk is a transaction then
2      result =
         isSound(txblk, view) ∧ isAuthenticated(txblk, numID, table, c_tail);
3  else
       // txblk is a block
4      if isAuthenticated(txblk, numID, table, c_tail) then
           // The result is initially set to true for an authenticated block, but its
              final value depends on the soundness and authenticity of each
              individual transaction inside the block
5          result = true;
6          for transaction tx ∈ txblk.S do
7              if
                 ¬isSound(tx, view) ∨ ¬isAuthenticated(tx, numID, table, c_tail)
                 then
8                  result = false;
9                  break;
10     if result == false then
           // Misbehavior detection (see Section 4.8)
11         find the guilty node and report it in cont;
           // Generate a misbehavior transaction
12         TXB-Generation(numID, table, c_tail, cont);
```

Algorithm 7: ViewUpdate

Input: numerical ID of the peer $numID$, routing table of the peer $table$, local view of the peer on blockchain's tail c_tail, view of the peer $view$, set of peer's collected new transactions $\mathcal{S}$

Output: updated $view$, updated $\mathcal{S}$

```
1  if view is empty then
       // An empty view corresponds to a new node that requires randomized
          bootstrapping to create its view
2      while i ∈ [1, α] ∧ less than t consistent views obtained do
           // Find the i^th view introducer
3          view_intro_i = H(numID||i) ;
4          search_proof_i = searchForNumericalID(view_intro_i, table);
5          if Verify(search_proof_i, numID, table, c_tail) then
6              contact i^th view introducer and obtain its view;
7      if t consistent views obtained then
8          update view;
9  else
       // Update the view towards the current tail of the blockchain using a
          search for name ID of the local view of the current tail within the overlay
10     search_proof_tail = searchForNameID(c_tail, {table});
11     if Verify(search_proof_tail, numID, table, c_tail) then
12         if set of new transactions {TX} ∈ search_proof_tail then
13             add tx ∈ {TX} with Audit(tx, view, numID, table, c_tail) == true
                 to S;
14             if min_tx new transactions are in S then
                   // Generate a block out of the collected transactions
15                 TXB-Generation(numID, table, c_tail, S);
16         else if new block(s) found then
17             if there is a fork then
18                 follow the block blk with minimum hash value;
               // Misbehavior verification
19             if Audit(blk, view, numID, table, c_tail) then
20                 update the view based on the new block;
21                 c_tail = blk;
                   // Dropping existing transaction pointers (See Section 4.7 for
                      more details)
22                 for tx ∈ blk.S do
23                     if holding a transaction pointer to tx.owner then
24                         drop the transaction pointer;
```

Algorithm 8: hasBalanceCompliance

Input: transaction tx, view of the validator $view$
Output: boolean $result$
// Retrieve the state of the transaction owner from the view of the validator peer
1 ($lastblk$, $state$, $balance$) = $view.get(tx.owner)$;
2 **if** *owner has enough balance to cover the routing and validation fees* **then**
3 | $result = true$;
4 **else**
5 | $result = false$;

Algorithm 9: Proof-of-Validation (PoV)

Input: transaction/block $txblk$, numerical ID of the validator peer $numID$, view of the validator $view$, local view of the validator on blockchain's tail c_tail
Output: message msg

```
1  if txblk is a transaction then
       // The validation of a transaction
       // Check the transaction's soundness, correctness, authenticity, and the
          balance compliance of its owner
2      msg = isSound(txblk, view) ∧ isCorrect(txblk, view) ∧
        isAuthenticated(txblk, numID, table, c_tail) ∧
        hasBalanceCompliance(txblk, view);
3  else
       // The validation of a block
       // Check the authenticity and consistency of the block
4      if isAuthenticated(txblk, numID, table, c_tail) ∧ txblk.prev == c_tail
        then
           // Check the soundness and authenticity of each individual transaction
              in the block
5          for tx ∈ txblk.S do
6              msg = isSound(tx, view) ∧ isAuthenticated(tx, numID, table,
                c_tail);
7              if more than one transaction from tx.owner ∈ txblk.S then
8                  msg = false;
9              if msg == false then
10                 break;
11     else
           // A consistency or authenticity violation found in the block
12         msg = false
13 if msg == true then
14     msg = signature on txblk.h by the validator's signing key;
       // Holding a replica of the validated transaction or block
15     insert txblk into the overlay as a node;
16     if txblk is a block then
17         if txblk is knocked-out in a fork then
18             drop txblk from the overlay;
19             terminate;
20         else
               // Adding transaction pointers for direct state retrieval (See
                  Section 4.7 for more details)
21             for tx ∈ txblk.S do
22                 insert a transaction pointer to tx.owner into the overlay;
23     if txblk is a transaction then
24         while txblk is not included in a block do
25             wait;
26         drop txblk from the overlay;
27 send msg to txblk.owner;
```