# Decentralizing Permissioned Blockchain with Delay Towers

***Abstract*—Growing excitement around permissionless blockchains is uncovering its latent scalability concerns. Permissioned blockchains offer high transactional throughput and low latencies while compromising decentralization. In the quest for a decentralized, scalable blockchain fabric, i.e., to offer the scalability of permissioned blockchain in a permissionless setting, we present L4L to encourage decentralization over the permissioned Libra network without compromising its sustainability. L4L employs delay towers, – puzzle towers that leverage verifiable delay functions – for establishing identity in a permissionless setting. Delay towers cannot be parallelized due to their sequential execution, making them an eco-friendly alternative. We also discuss methodologies to replace validators participating in consensus to promote compliant behavior. Our evaluations found that the cost of enabling decentralization over permissioned networks is almost negligible. Furthermore, delay towers offer an alternative to existing permissionless consensus mechanisms without requiring airdrops or pre-sale of tokens.**

***Index Terms*—Blockchain, centralization/decentralization, delay towers, verifiable delay function, system design and analysis**

## I. Introduction

Blockchains constitute distributed ledgers, that overcome the challenges of centralization, for the exchange of digital assets without trusted intermediaries in decentralized and a priori trustless environments [1]. However, the promise of decentralization poses the challenge of needing to reach consensus on a single immutable source of truth among untrusted parties, and not everyone might abide by the rules of the protocol. Without loss of generality, it is fair to assume a small percentage of (un)intentional malicious parties that act in ways that hinder the integrity and security of the blockchain network. Consequently, the consensus protocols backing blockchains must tolerate Byzantine failures in addition to benign failures. More specifically, blockchains are also prone to Sybil attacks wherein a malicious party subverts the consensus protocol by creating a large number of (pseudo)anonymous identities [2]. During a Sybil attack, the malicious node can prevent creating new blocks or manipulate the ordering of transactions. To enable Sybil resistance and ensure security, blockchain protocols must establish and persist identities.

**Classification of Blockchains.** Based on how blockchains establish identity and build Sybil-resistance, we classify blockchains into two broad categories – permissioned and permissionless. Permissioned blockchains such as Hyperledger Fabric [3] and Libra[1] [4] rely on a membership service provider (MSP), a trusted entity, to certify and authorize nodes to access the network. This MSP moderates the network and can revoke authorized access for malicious nodes involved in either perceived Byzantine behavior or Sybil attacks. On the other hand, permissionless blockchains, also often referred to as public blockchains, are more decentralized, i.e., they do not have a centralized authority to manage access to the network. To establish identity, permissionless blockchains such as Bitcoin [1] and Ethereum [5] leverage proof of work (PoW), wherein the nodes compute hash puzzles. The computational power required for PoW serves as a proxy for identity to enable Sybil resistance. The nodes with high computational resources would have a higher probability of proposing the next block to be appended on the decentralized ledger. The nodes in the network reach a consensus on the proposed block if there are no conflicts. In case of conflict, the longest chain is chosen as the valid ledger [1]. It is worth noting that block proposal time is different from time to reach finality (confirmation). For instance, in Bitcoin, a probabilistic agreement is used to reach finality; it takes approximately an hour to reach finality after a block proposal [1], [6]. Unlike permissionless blockchains with PoW, permissioned blockchains tend to use Byzantine fault-tolerant (BFT) protocols to reach consensus [3], [4]. BFT protocols have deterministic finality and tend to reach finality faster, i.e., when a quorum of nodes agrees upon a proposed block, and they offer better scalability than PoW [7]. We consider a system to be more scalable if it offers higher throughput (transactions/second) and faster finality (transaction confirmation time). For comparison, Bitcoin and Hyperledger have a throughput of around seven and a few thousand transactions per second, respectively [8]. An overhead exists to build Sybil resistance and enable decentralization, leading to a trade-off between decentralization and scalability of blockchains [9]–[12]. An increase in decentralization often resulted in lower scalability, i.e., low throughput and high finality time, see Table. I.

TABLE I
Comparison of distributed ledger technologies

| | **Hyperledger or Libra** | **Bitcoin or Ethereum** |
|---|---|---|
| **Type** | Permissioned | Permissionless |
| **Consensus** | BFT | PoW |
| **Throughput** | High | Low |
| **Finality** | Fast | Slow |
| **Decentralization** | Low | High |

**The Quest.** The problem this paper addresses is characterizing a blockchain that encourages decentralization without

[1]On December 1st, 2020, the Libra blockchain was renamed Diem blockchain.

compromising scalability; in other words, achieving consensus without trusted intermediaries at scale. This problem is considered in the literature as the quest for the scalable blockchain fabric (e.g., [13]). This quest for decentralized, scalable consensus is uplifting because successful expeditions could take us one step closer to the massive adoption of blockchains.

**Expeditions.** There are many ongoing expeditions for this quest (e.g., [12], [14], [15]). Broadly, we can achieve decentralized scalable consensus by either scaling permissionless blockchains or decentralizing permissioned blockchains. Lately, multiple on-chain and off-chain solutions to scale permissionless blockchains are being developed, such as sharding [16], [17], zero-knowledge and optimistic rollups [18]–[20], side chains [21], bidirectional payment channels [22], and directed acyclic graph-based ledgers [23], [24].

Here, we explore the latter approach of decentralizing permissioned blockchains. We choose to extend BFT protocol-based blockchains to decentralized settings because they better address scalability challenges, and we only need to introduce decentralization. Though this sounds straightforward, making a BFT protocol permissionless raises a plethora of challenges. Firstly, a malicious party could join the network without a membership service provider and create many anonymous identities to execute a Sybil attack. Secondly, BFT protocols need a defined set of nodes, and they have to be defined without a trusted intermediary. Having failed nodes would increase the time to reach consensus; for instance, if a crashed node is chosen to propose a block, it results in a timeout instead of a new block [25]. Also, BFT protocols do not scale-out, i.e., with the increase in the number of nodes, lower throughput, and higher latency were recorded [7]. Thirdly, incentive mechanisms are needed to reward honest behavior and punish bad behavior for not contributing to the protocol [25], [26].

**L4L Protocol.** In this paper, we describe L4L to decentralize a permissioned blockchain that uses a BFT protocol for consensus. Specifically, we enable decentralization over the permissioned Libra blockchain [27]. We believe that a protocol should not have a centralized MSP to be decentralized. To achieve decentralization without MSP, we must establish identities in a permissionless setting. To build persistent identities without heavy PoW computations and presale of native tokens, we introduce *delay towers*. Delay towers are puzzle towers [28] that act as a *proof of elapsed time* (PoET) leveraging verifiable delay functions (VDF) [29]. A VDF is a cryptographic delay function that is more eco-friendly than PoW because it cannot be parallelized, with no substantial benefit in using more computational resources. L4L uses VDFs for building delay towers. Each node locally executes a VDF to generate proofs at regular intervals, and these proofs are chained to build delay towers, i.e., each proof executes from the hash of the previous proof. In addition, L4L reconfigures nodes participating in consensus at regular intervals by punishing unwanted behavior, for example, eliminating failed nodes. The nodes which participate actively, i.e., by participating in PoET and attesting blocks to reach consensus, are given preference to validate transactions during reconfiguration. Though this approach might have a few similarities to proof of stake (PoS) as in Tendermint [30] or Algorand [31] due to usage of a BFT protocol, they are not the same because L4L relies on delay towers and not staking of native assets to establish identities. Consequently, L4L can be a fairer alternative to airdrop or pre-sale of native tokens for bootstrapping blockchain networks.

**Contributions.** The key goal of L4L is decentralizing a permissioned blockchain. On this journey, L4L contributes the following principle ideas:

1) Delay towers to encourage persistent identities in permissionless settings;
2) Mechanisms to reconfigure nodes participating in consensus to engender compliant behavior;
3) More eco-friendly and sustainable alternatives to existing consensus protocols in permissionless settings;
4) Mechanisms to bootstrap a blockchain network without token sales, airdrops, or susceptibility to mining attacks.

**Outline.** The paper is organized as follows. The next section introduces blockchain basics, VDFs, and the Libra blockchain. Having discussed how to construct VDFs, §III addresses ways of establishing identity using delay towers in L4L. This section further describes the reconfiguration of the nodes participating in the BFT consensus at regular intervals in L4L. We evaluate the cost of decentralization using delay towers from the perspective of both the node and network in §IV. Finally, we conclude with future directions in §V.

## II. Background

In this section. we introduce some basic blockchain formalism and introduce verifiable delay functions (VDF), the building blocks of delay towers; both of which are used in subsequent parts of this paper. We end this section with an overview of the permissioned Libra blockchain.

### A. Blockchain Notation

The formalism we introduce here, serves us to characterize the L4L blockchain that is based on a BFT consensus protocol; the same formalism may not readily apply to other blockchain models. A blockchain is comprised of a distributed ledger $L$ composed of blocks $B_i$, represented by $L_n = \{B_0, B_1, B_2, ....., B_n\}$. The genesis block is the first block on the ledger and is denoted by $B_0$. Each block $B_{k+1}$, except the genesis block $B_0$, extends the current state of the ledger $L_k$ by appending the hash of the latest committed block, $hash(B_k)$. The memory pool, denoted as $mem$, stores the list of transactions that are not yet published onto the ledger, represented by $mem = \{tx_j | tx_j \notin L\}$. The block proposers pick a set of transactions $tx_j \in mem$ from the memory pool to create blocks as follows $B_i = \{tx_0, tx_1, tx_2, ...tx_q\}$. A block $B_{k+1}$ is valid if and only if every transaction in that block $B_{k+1}$ is valid, and the block contains the hash of the latest committed block $B_k$.

Let $U = \{u_1, u_2, ..., u_w\}$ be the users of this blockchain network. Each user $u_i$ has a public-private key pair $\{pk_i, sec_i\}$.

A message $msg$ endorsed by user $u_i$, using private key $sec_i$, is represented as $sig_i(msg)$ and one can use the public key $pk_i$ to check if $sig_i(msg)$ is indeed signed by $u_i$.

Let $N = \{n_1, n_2, ...\}$ be the set of nodes that are interested in listening to the blocks $B_k$ on the network. The full nodes $f \subseteq N$ are the subset of nodes that do listen to blocks $B_k$ on the network to replicate the ledger $L_k$ locally, and they act as auditors to the ledger fostering decentralization. To summarize, nodes become full nodes by initializing and tracking the evolving ledger state.

The process of running delay towers is called PoET mining, and the nodes that PoET mine are called miners. Note that mining in this paper does not refer to PoW-mining. Furthermore, mining in itself is not adding utility to the blockchain network, as mining does not always mean participating in the consensus protocol but means a miner is building a persistent identity in the blockchain.

The validator set consists of nodes that participate in the consensus protocol, i.e., validate the correctness of blocks, represented by $VS = \{v_1, v_2, ..., v_t\}$ and $VS \subseteq f \subseteq N$. For instance, if a sender of a transaction uses an incorrect signature or does not have the required minimum account balance, the block containing that transaction is not valid. Since BFT protocols generally require $3f + 1$ validators where up to $f$ of them could be faulty, we assume that there are at least 4 validators, i.e., cardinality of validator set is always greater than or equal to four $|VS| \geq 4$. A validator quorum $\mathbb{Q}$ is attained if a block is endorsed by the ceiling of at least two-thirds of the validator set, $|\mathbb{Q}| \geq \lceil (2/3) * |VS| \rceil$.

We define a loosely synchronized clock function $t$ based on UNIX epoch timestamps. After each fixed interval $t'$ in $t$, when $t \bmod t' = 0$, we have a new epoch $E$, represented by a natural number. In other words, an epoch refers to a fixed time interval. The validator set *VS* remains constant for the duration of an epoch; however, it can be different for different epochs. The concept of epochs helps the protocol to reset its validator set *VS* by replacing the failed validators at regular intervals. In the current trails of L4L, the length of an epoch is set to a day (24 hours).

### B. Verifiable Delay Functions

We use verifiable delay functions (VDFs) as a proof of elapsed time (PoET) to establish persistent identities. A VDF is a cryptographic construct that intuitively speaking slows things down. A VDF takes a specified number of sequential steps to be evaluated. Here, we go through the formal specification and properties of VDFs as defined by Boneh et al. [29]. A VDF is a set of three functions, `VDF=(setup,eval,verify)`.

1) `setup(security,t)` $\rightarrow$ `pp`. the setup takes in security parameters $security$ including pre-defined puzzle difficulty, and time-bound `t` to output the public parameters `pp`. These public parameters are unique to each participant and they define the input space $X$ and output space $Y$.
2) `eval(pp,x)` $\rightarrow$ `(y,p)`. this is a deterministic function that takes time-bound `t` sequential steps to compute ($f: X \mapsto Y$). This function consumes the public parameters *pp* and an input $x$ from the input space `x` $\in X$ to output the computation result in the output space `y` $\in Y$ and the proof of computation `p`. This is the delay component in the VDF.
3) `verify(pp,x,y,p)` $\rightarrow$ $\{$`true,false`$\}$. this is a Boolean function to verify the correctness of the output `y` and proof `p`.

The `eval` function of VDF adheres to two important properties - uniqueness and sequentiality.

1) Uniqueness: $\forall x \in X, \exists y \in Y$:$f$(`pp, x`)=(`y, p`). For every defined input `x`, the output of the `eval` function `y` is unique and deterministic. However, the proof $p$ generated during evaluation might vary.
2) Sequentiality: The `eval` function takes at least time-bound `t` steps to compute, irrespective of parallelization capabilities. In other words, any parallel or random algorithm cannot estimate the output in less than `t` steps.

Having defined the properties of the `eval` function, we study candidate solutions. Time-lock puzzles are slow functions that involve computing an inherently sequential function [32]. Boneh et al. [29] proposed a generalization of time-lock puzzles as a candidate for the `eval` function. The computation of repeated squaring in a group of unknown order would take `t` steps, even on a machine with parallelization capabilities [33]. This function is as follows:

$$f(x) = x^{2^t} \mod \mathbb{N} \tag{1}$$

The final step in the VDF construction is to quickly verify the correctness of output and proofs. The candidates for `verify` were presented by Wesolowski [34] and Pietrzak [35]. We choose the Pietrzak scheme for our VDF due to its performance benefits in the verification step [36].

L4L realizes chaining of VDF proofs to build delay towers to form persistent identities (see §III-A for details).

### C. Libra Blockchain

*Libra* is a permissioned blockchain that has numerous advantages. Firstly, Libra uses LibraBFT, a leader-based BFT protocol under partial synchrony assumptions that guarantees safety and liveness with deterministic finality [4]. LibraBFT relies on a quorum $\mathbb{Q}$ for consensus and, as a result, prefers safety over block production (liveness) [7]. LibraBFT builds on HotStuff, offering linear communication complexity for reaching consensus, making the throughput depend only on the network latency [7]. In fact, HotStuff is reported to offer better scalability (higher throughput and lower latency) [37] than PBFT [38], Tendermint [30], and Streamlet [39]. Pipelining in HotStuff enables the finality of a proposed block by the third block following the proposed block.

Secondly, Libra uses the Move programming language for smart contracts. The Move language offers security and formal verifiability needed for smart contracts [40], [41]. Libra also

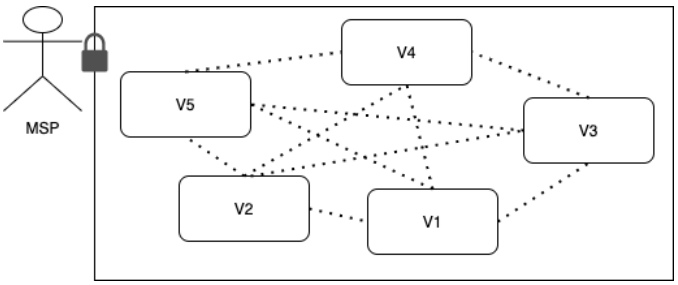


Fig. 1. Membership Service Provider (MSP) decides on the validator set (VS) in permissioned blockchain such as Libra

has modules for network synchronization, storage, and cryptographic primitives, among others. Finally, the Libra project is open-source, hosted on GitHub under the Apache 2.0 license. L4L aims to decentralize the Libra blockchain to provide these benefits in a permissionless setting.

## III. L4L: Design and Implementation

L4L is a consensus protocol for permissionless blockchains designed to foster decentralization, scalability, and sustainability. In other words, L4L is a Sybil-resistant Byzantine fault-tolerant state machine replication protocol developed by decentralizing the general-purpose permissioned Libra blockchain without compromising sustainability.

A criticism of permissioned blockchains is the lack of decentralization as access to the network is controlled. Nonetheless, few blockchains reasonably assert that permissioned blockchains are decentralized as their transaction processing is decentralized, i.e., no single validator can decide on a state transition of ledger $L$ by itself, and consensus requires a quorum $\mathbb{Q}$. However, this view fails to acknowledge the presence of a centralized authority, the MSP, who chooses the validator set. Having an MSP administrating the network's validators could inherit the biases of MSP in choosing validators and could potentially result in a biased quorum $\mathbb{Q}$ assembled by an (un)trusted intermediary. Moreover, the MSP is a point of centralization, requiring a trusted intermediary and consequently, causing the protocol to lose decentralization, see Fig. 1.

The goal of L4L is to *liberate the permissioned Libra blockchain* from its decentralization-inhibiting MSP to render it permissionless. Thus, there is a need to eliminate the requirement for a trusted intermediary, the MSP. However, a BFT-based network needs a fixed set of validators to guarantee liveness and safety. An MSP plays a crucial role in effectively administering the validators in the network, i.e., the MSP approves requests for joining the validator set *VS* using the identity of that node in the real world. Perhaps the most severe disadvantage of eliminating the MSP is that there is no identity associated with candidates in the validator set, and this identity has to be established by other means in a permissionless setting.

Let us review the existing practices in establishing identity (building Sybil resistance) in permissionless blockchains. PoW

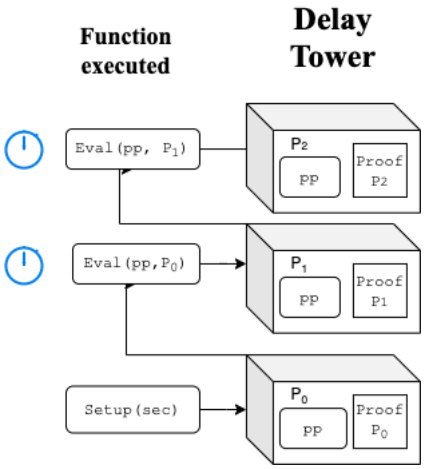


Fig. 2. A delay tower is a series of VDF proofs

blockchains use computational power as a proxy for identity, raising concerns about eco-friendliness [42], [43]. Another prominent approach is proof of stake (PoS), wherein the validators stake in their assets as native tokens to establish identity. Nevertheless, this approach requires token distribution such as initial coin offerings (ICOs) or airdrops. There is a need for establishing identity in a permissionless setting, and delay towers could be a promising alternative to existing approaches.

### A. Delay Towers as Proof of Elapsed Time

To establish the persistent identities required for BFT-based networks, L4L introduces the concepts of delay towers. Drawing inspiration from Sybil-resistant network identities from dedicated hardware in [28], delay towers extend the notion of puzzle towers with VDFs. The usage of VDFs addresses the sustainability challenges of PoW puzzles, such as susceptibility to mining attacks or environmental concerns.

Delay towers are a sequential series of sequential proofs. The process of building delay towers is called *PoET mining*, and every node that is mining is called a *miner*. All the miners form the miner pool $M \subseteq N$. Unlike PoW algorithms that are parallelizable and probabilistic, PoET mining is sequential and deterministic. Since VDFs cannot be parallelized, they have no substantial benefit in better hardware such as GPUs [29]. Each proof extends from the previous proof to build the tower, creating sequential series of sequential work. Delay towers enable persistent identities by providing permissionless and non-forgeable identities with minimal capital. Delay towers act as proof of elapsed time (PoET), and the height of the delay tower denotes how long the miner has been participating in the network, esentially showing commitment to the network and providing a metric that can be used to rank candidates for inclusion into the validator set *VS* in L4L.

Every miner $m$ in L4L has a delay tower, represented by $T_m = \{P_0, P_1, P_2, ....\}$. Each proof $P_{n+1}$, except $P_0$, builds from its parent proof $P_n$, proving the work done by

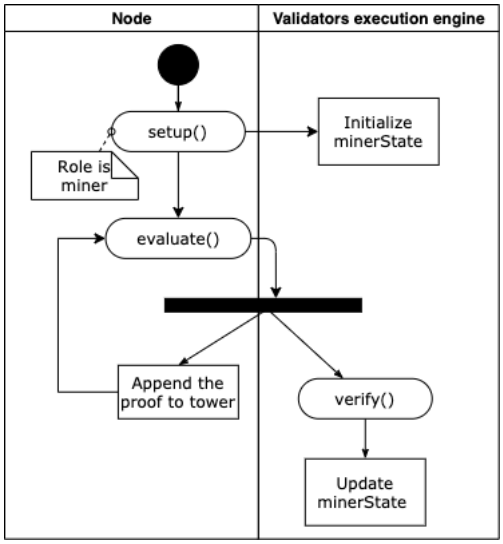


Fig. 3. Activity diagram of miner

**Algorithm 1:** validVDFProof(addr, proof)

**Output:** true/false

```
// Authorization check
result ← false
if validSignature(addr, proof) then
    // Obtain the miner state
    ms ← getMinerState(addr)
    // Preliminary Checks
    if ms.hash == proof.previous_proof_hash and
     ms.height <proof.height then
        // Verify the VDF correctness- Pietrzak
        if VDF.verify(proof) then
            ms.height ← ms.height + 1
            ms.num ← ms.num + 1
            ms.hash ← computeHash(proof)
            result ← true
return result
```

the miner after its newest proof $P_n$ as shown in Fig. 2. In terms of implementation, `setup` and `evaluate` are executed locally in the miner node, and the proofs are sent on-chain for verification wherein validators execute the `verify` function, as shown in Fig. 3. The security parameters `sec`, which are an input for `setup`, are fixed during the genesis of the network.

A full node $f \in N$ becomes a miner by running `setup` and submitting its first VDF proof to initialize its *miner state* as shown in Fig. 3. The `setup` function receives security parameters `sec` and public parameters `pp` as input to produce their first VDF proof. Public parameters include the cryptographic public key $pk_f$ and the IP address of the dedicated hardware. This step links the public key with the delay tower, making the delay towers non-transferable. The output from `setup` is the signed parameters ($sig_f(params)$) which are submitted to the memory pool ($mem$) from where validators check if they constitute a valid proof to update the ledger state then ultimately. Valid proof is defined as follows.

**Definition III.1.** VALID PROOF. A VDF Proof $P_{k+1}$ is valid if it is either the first proof $P_0$ with initialization parameters submitted by the full nodes or a proof that builds on top of the hash of the latest proof $hash(P_k)$. Moreover, a given proof returns `true` when input to the `verify` function.

If the proof $P_0$ generated from the `setup` is valid, the miner state of that full node gets initialized, after which the full node officially becomes a miner, joining the miner pool *M*. Each miner $m$ in the miner pool, $\{m|m \in M\}$, performs PoET mining, i.e., iteratively executes `eval` locally, to submit VDF proofs to the validator set *VS*. Each proof computation, `eval`, takes `t` steps using the hash of the previous proof as an input to generate the output proof. To summarize, a node executes `setup` that produces the foundation of its delay tower and becomes a miner to execute `eval` to keep growing its tower (see Fig. 2).

Thus far, we have discussed the miner $m$ view of building delay towers $T_m$ via PoET mining. We now shift our focus to what happens on the validators' end, who participate in the consensus process. When a full node submits the output from executing `setup` locally, the *miner state* of that node is initialized as its role changes from a full node to a miner. The miner state $ms$ stores the tower height denoted by $height$, the hash of the latest verified proof as $hash$, the number of proofs submitted in the current epoch as $num$, $jailbit$ to signify whether the miner is jailed, and the remaining sentence $jail\_sentence$ if it is jailed. We discuss jailing in the following.

For all the sequential proofs $P_{k+1}$ submitted by the miner, the validators use Algorithm 1 that runs in O(1). The validators *VS* begin by checking authorization, i.e., the validity of the signature submitted by miner $m$. The validators then check if the proof submitted uses the last verified proof $hash(P_k)$ and if the tower height tallies with $height$ stored on-chain. Finally, the validators execute the `verify` function to verify the correctness of VDF proofs using the Pietzak scheme (see §II-B). If proven correct, the miner state of the sender is updated, i.e., the validator increments the number of proofs submitted in epoch, $num$, and the tower height, $height$. The validator also updates the hash of the last verified proof $hash$ to $hash(P_{k+1})$, and this process repeats for the duration of PoET mining.

### B. Reconfiguration of Validator Set

Delay towers build a persistent identity that proves PoET, which is non-transferable, takes time to acquire, and requires minimal resources. However, the design described thus far fails to acknowledge the presence of failed nodes due to crash failures or, even worse, Byzantine behavior. If a failed validator is responsible for proposing a new block $B_{m+1}$, the protocol would not be able to give rise to a new block. Instead, the network at $L_m$ observes a timeout and increases in latency

or decreases in throughput. Even worse, if more than one-third of validators fail, the liveness of L4L is compromised due to lack of achieving a quorum $\mathbb{Q}$. Thus, the protocol needs mechanisms to punish bad behavior for keeping the system intact.

To address these challenges, L4L reconfigures the validator set *VS* at regular intervals, that is, at the end of each epoch $E$, to jail failed or non-conformat validators and to generate a new validator set.

*1) Jail inactive validators:* As explained earlier, it is clear that failed validators do more harm than good by affecting the network's performance, and there is a need to punish bad behavior to maintain the integrity of the protocol [9], [26]. Essentially, the protocol has to answer how to measure bad behavior and punish it? To do so, we introduce *liveliness*, a metric to measure how actively a validator node is contributing to the consensus.

**Definition III.2.** LIVELINESS. The liveliness of a validator $v_i$ is the ratio of the number of appended blocks signed by a validator to the total number of blocks appended to the ledger $L$ in the given epoch. Liveliness in an epoch $E$ is denoted by $l_E$.

$$l_E(v_i) = \frac{\text{blocks signed by } v_i}{\text{total blocks}} \tag{2}$$

At the end of every epoch, L4L measures the liveliness of all the validators. All the validators that do not meet the liveliness threshold $\pi$ are considered to have failed. L4L punishes the failed validators by jailing them, i.e., failed validators lose their role as a validator for the upcoming epoch and their role is reverted back to miners, and they go to jail *J* for jail sentence $\psi$, measured in number of epochs $E$. To get out of jail, the jailed miners should be PoET mining a threshold $\mu$ number of proofs every epoch for the period of the jail sentence, set as a global constant, $\psi$. Algorithm 2 takes time O($n$) to jail failed validators where $n$ is the cardinality of the validator set $|VS|$. The $jailbit$ and $jail\_sentence$ are variables of the miner state.

**Algorithm 2:** jailFailedValidators( )

```
foreach val in VS do
    ms ← getMinerState(val)
    // Check if threshold liveliness is met
    if L_E(val) ≤ π then
        ms.jailbit ← 0
        ms.jail_sentence ← ψ
```

*2) Select the top* $\mathbb{N}$ *validators:* Now that we jailed the failed validators, the next step is to choose the validator set from the candidate pool called validator universe *VU*. $VU$ is a subset of nodes from the miner pool ($VU \subseteq M$) who state their interest in becoming validators in the next epoch $E+1$ by mining the threshold $\mu$ number of proofs in the current epoch $E$, stored as $num$ in the miner state. The extraction of nodes from the validator universe *VU* involves excluding jailed miners and resetting the number of proofs $num$ submitted in the epoch to zero. This step takes linear time in the cardinality of the miner pool *M* as shown in Algorithm 3.

**Algorithm 3:** getValidatorUniverse( )

```
Output: VU
Data: M
VU ← ∅
foreach miner in M do
    ms ← getMinerState(miner)
    if ms.num > μ & miner ∉ J then
        VU.insert(miner)
    ms.num ← 0
```

Although all candidates in the validator universe *VU* are compliant, having everyone in the validator set is not ideal due to the limitations of BFT protocols. BFT protocols do not scale-out, i.e., with the increase in the cardinality of the validator set, throughput drops, and latency increases [7], [16]. To counter these challenges, L4L caps the cardinality of the validator set *VS* to $\mathbb{N}$, $|VS| \leq \mathbb{N}$ and fills these slots by non-failed validators. There is a tradeoff here: having a high $\mathbb{N}$ leads to scalability challenges, and having a too low $\mathbb{N}$ leads to loss of decentralization. In the current implementation, $\mathbb{N}$ is set to a hundred based on previous experimental results [7].

Since the candidates in the validator universe *VU* might be more than the cardinality of the validator set $\mathbb{N}$, we need a mechanism to select candidates; here, a ranking mechanism is used. L4L uses the tower height to rank the candidates due to various advantages. Firstly, tower height is an easy to compute and deterministic metric to measure how long the miner has been actively PoET mining. Secondly, it is linear, providing only a minimal advantage to genesis nodes compared to PoS mechanisms. Moreover, building delay towers is permissionless, requiring minimal capital. A new validator set is proposed using Algorithm 4 that takes time linear in the order of *VU*.

**Algorithm 4:** proposeValidatorSet( )

```
Output: VS
Data: M
VS ← ∅; i ← 0
VU ← getValidatorUniverse()
sort VU on w
// Select top ℕ validator candidates
while i<ℕ and i<len(VU) do
    val ← VU[i]
    VS.insert(val)
    i ← i + 1
```

With the output from the above algorithm, L4L reconfigures the validator set $VS_E$ to the proposed validator set $VS_{E+1}$. On successful reconfiguration, the new epoch $E+1$ begins

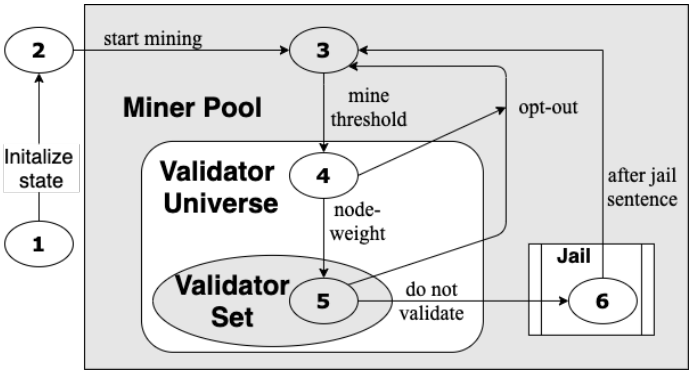


Fig. 4. The lifecycle of a node

TABLE II
STATES OF THE NODES IN THE L4L PROTOCOL

| State | Role | Set | PoET Mining | BFT Validating |
|---|---|---|---|---|
| 1 | Node | *N* | OFF | N/A |
| 2 | Full Node | *N* | OFF | N/A |
| 3 | Miner | *M* | ON/OFF | N/A |
| 4 | Miner | *VU* | ON | N/A |
| 5 | Validator | *VS* | ON | ON |
| 6 | Miner | *J* | ON/OFF | N/A |

with the updated validator set $VS_{E+1}$ without any downtime imposed on the network.

### C. Lifecycle of a Node

This section summarizes the L4L protocol by going through a node's lifecycle and responsibilities as depicted in Fig. 4 and Table II. Full nodes $f \subseteq N$ refer to the subset of nodes $N$ that initialize their ledger $L$ and listen to incoming blocks, transitioning from State 1 to 2 in Fig. 4. All full nodes that execute `setup` become miners and enter the miner pool $M$, transitioning from State 2 to 3 in Fig. 4. The miners who mine a threshold $\pi$ number of proofs in an epoch $E$ are stating their interest to become validators in the following epoch *E+1*. These miners are candidates for the validator set for the following epoch $VS_{E+1}$; they form the validator universe $VU \subset M$, represented by State 4. The validator set ($VS \subseteq VU$), represented by State 5, is chosen by ranking the miners in the validator universe $VU$. During reconfiguration, i.e., at the end of the epoch, the failed validators who do not meet the liveliness threshold $\pi$ are sent to jail $J$ as captured by State 6, and the validators who do not meet the PoET mining threshold $\mu$ return to become miners as captured with State 3. To get out of the jail, i.e., the transition from State 6 to State 3, the miners have to PoET mine for $\psi$ jail sentences.

## IV. EVALUATIONS

L4L uses delay towers to enable permissionless blockchains. The delay towers use Chia's Pietzak VDF implementation with 120 million iterations and 512 bits for the length of prime numbers generated in the proof. This profile has an estimated lower bound of thirty minutes for generating each proof.

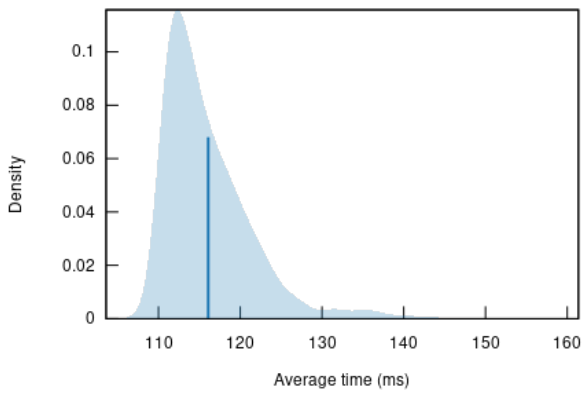


Fig. 5. Probability distribution of execution times of 1500 `verify` iterations, the line represents the mean

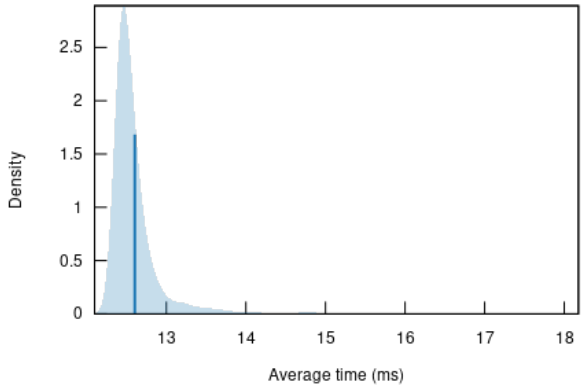


Fig. 6. Probability distribution of execution times of 1500 `verify` iterations for invalid proofs, line represents the mean

The miners `evaluate` proofs locally whereas the validators `verify` the validity of submitted proofs. The time employed for verifying the validity of proofs could instead be utilized for the client's transaction processing; it is the cost L4L pays for decentralization. We benchmark the `verify` function's execution time using 1500 samples, see probability distribution in Fig. 5 with a mean of 115.80 ms and a median of 114.56 ms.

If the length of the prime number generated as part of the proof deviates from 512 bits, the proof is rejected. Verification of invalid proofs of correct length has a mean of 12.6 ms (11% of valid proof computation) for 1500 samples as shown in Fig. 6. Assuming a validator submits 48 valid proofs (every thirty minutes) in an epoch, the overhead for verification is 5.52 seconds per validator per epoch. Extrapolating this to the entire validator set ($\mathbb{N} = 100$) would result in 552 seconds in an epoch (0.0063%). We claim that the overhead for enabling decentralization is minimal.

We now turn to the experiments on mining the delay towers by analyzing the time taken by the `evaluate` method. We

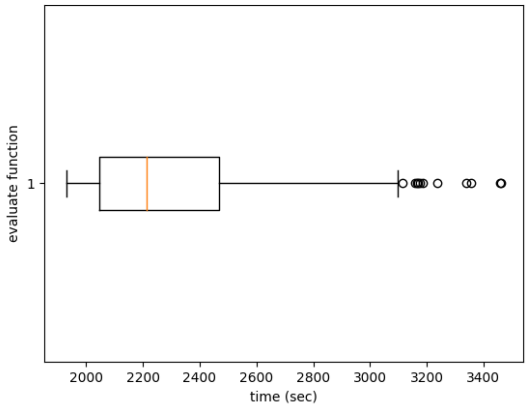


Fig. 7. Box plot of 1790 samples of execution times of `evaluate` function

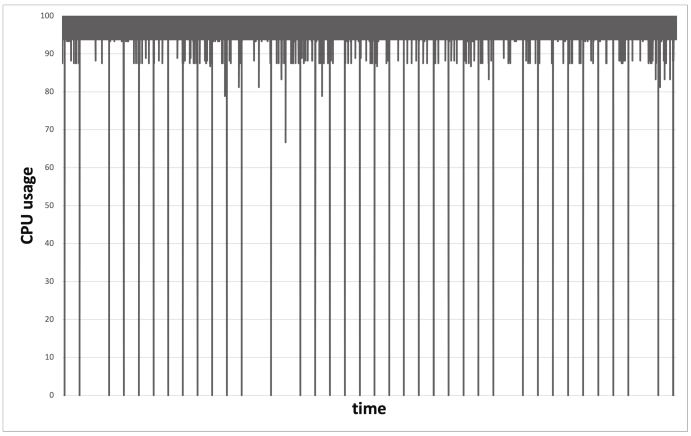


Fig. 8. CPU utilization by delay tower plotted for an epoch

collected data from a Linux octa-core instance with 16 GB RAM (Intel DO-Regular model). The mean and median for 1790 samples are 2362 s and 2310 s, respectively, as shown in Fig. 7. The $25^{th}$ and $75^{th}$ percentile are at 2045 s and 2467 s, respectively. The range is wide from 1929 s to 3463 s, and subtle differences such as the compiler version could affect the execution time. Furthermore, these insignificant benefits in individual VDF proof computation times compound to significant tower height over time.

The VDF computation plays a vital role in the security of the network. One could impose software and hardware uniformity to address inconsistencies, requiring every miner to use the same configurations; however, this requirement could increase costs and be a challenge to police without affecting the network's decentralization. Furthermore, trusted computing is more vulnerable than PoW [44]. As a countermeasure, L4L sets an upper limit on the tower height growth in an epoch to mitigate any variations different hardware may impose on VDF proof computations. The authors recommend using the number of epochs that a miner has mined a threshold $\mu$ of proofs, as an alternative to pure tower height, to select the top $\mathbb{N}$ validators. This alternative would ensure all the miners who meet a threshold to be considered equal irrespective of their CPU speeds.

As previously stated, L4L uses delay towers to provide decentralization without loss of sustainability. We measure the computational resources of running delay towers to understand the ecological impact. Using the same Linux instance as mentioned earlier, we collected the CPU consumption for building a delay tower over an epoch, as shown in the Fig. 8. The delay tower uses one CPU core instance in an octa-core processor, and there is a drop in CPU utilization between the proofs. Because the process cannot be parallelized, no more than one CPU core can ever build the same delay tower. The delay towers use minimal system memory or network for their processes. The data show that delay towers add minimal overhead to both the nodes and the network.

## V. Conclusions

Intending to leverage the scalability benefits of permissioned blockchains in permissionless settings, L4L decentralizes the permissioned BFT-based Libra blockchain. To establish identities required for BFT networks, L4L utilizes delay towers that are puzzle towers with VDFs. Delay towers offer various benefits, including lowering the barriers to entry with minimal capital requirements; anyone with a CPU can mine delay towers. Importantly, PoET mining delay towers are environmentally sustainable as they use minimal computational resources, i.e., its sequential nature ensures no advantage from parallelization. Since the growth of a delay tower is linear, the advantage of peers in genesis decreases over time and provides an opportunity to anyone interested. Furthermore, delay towers enable bootstrapping a network without token sales, airdrops, or being susceptible to mining attacks.

With the limitations of the BFT network, we showed how network could scale from permissioned settings to permissionless setting. Let us measure the degree of decentralization using the Nakamoto coefficient. The Nakamoto coefficient is the minimum number of validators needed to collude to compromise the system, i.e., affect the safety or liveness of the network [45]. In a decentralized L4L network of hundred validators with no Sybils, 33 validators need to collude to affect the liveness because of the two-thirds requirement for a quorum $\mathbb{Q}$ and, hence, the Nakamoto coefficient for L4L liveness is 33. For reference, the Nakamoto coefficient for safety in Bitcoin and Ethereum is less than five [46]. Furthermore, it would be interesting to draw ideas from PoS approaches to design delegation and tower stacking as a mechanism to scale participation in consensus beyond the limits of BFT.

## References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Manubot, Tech. Rep., 2019.

[2] J. R. Douceur, "The sybil attack," in *International workshop on peer-to-peer systems*. Springer, 2002, pp. 251–260.

[3] 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 *Proceedings of the thirteenth EuroSys conference*, 2018, pp. 1–15.

[4] M. Baudet, A. Ching, A. Chursin, G. Danezis, F. Garillot, Z. Li, D. Malkhi, O. Naor, D. Perelman, and A. Sonnino, "State machine replication in the libra blockchain," *The Libra Assn., Tech. Rep*, 2019.

[5] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[6] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2015, pp. 281–310.

[7] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus with linearity and responsiveness," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019, pp. 347–356.

[8] J. Sousa, A. Bessani, and M. Vukolic, "A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform," in *2018 48th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*. IEEE, 2018, pp. 51–58.

[9] K. Zhang and H.-A. Jacobsen, "Towards dependable, scalable, and pervasive distributed ledgers with blockchains." in *ICDCS*, 2018, pp. 1337–1346.

[10] M. Bez, G. Fornari, and T. Vardanega, "The scalability challenge of ethereum: An initial quantitative analysis," in *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE, 2019, pp. 167–176.

[11] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, and Y. Liu, "A survey on the scalability of blockchain systems," *IEEE Network*, vol. 33, no. 5, pp. 166–173, 2019.

[12] A. Chauhan, O. P. Malviya, M. Verma, and T. S. Mor, "Blockchain and scalability," in *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 2018, pp. 122–128.

[13] 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, pp. 112–125.

[14] Q. Zhou, H. Huang, Z. Zheng, and J. Bian, "Solutions to scalability of blockchain: A survey," *IEEE Access*, vol. 8, pp. 16 440–16 455, 2020.

[15] S. Kim, Y. Kwon, and S. Cho, "A survey of scalability solutions on blockchain," in *2018 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 2018, pp. 1204–1207.

[16] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 583–598.

[17] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 931–948.

[18] P. Biel, S. Zhang, and H. Jacobsen, "A zero-knowledge proof system for OpenLibra," in *Middleware '21: Demos and Posters*. ACM, 2021, pp. 3–4.

[19] V. Buterin, "The dawn of hybrid layer 2 protocols," August 2019. [Online]. Available: https://vitalik.ca/general/2019/08/28/hybrid_layer_2.html

[20] H. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten, "Arbitrum: Scalable, private smart contracts," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 1353–1370.

[21] J. Poon and V. Buterin, "Plasma: Scalable autonomous smart contracts," *White paper*, pp. 1–47, 2017.

[22] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016.

[23] C. Li, P. Li, D. Zhou, Z. Yang, M. Wu, G. Yang, W. Xu, F. Long, and A. C.-C. Yao, "A decentralized blockchain with high throughput and fast confirmation," in *2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20)*, 2020, pp. 515–528.

[24] R. Alexander, *Iota-introduction to the tangle technology: Everything you need to know about the revolutionary blockchain alternative*. Independently published, 2018.

[25] G. Zhang and H.-A. Jacobsen, "Prosecutor: an efficient bft consensus algorithm with behavior-aware penalization against byzantine attacks," in *Proceedings of the 22nd International Middleware Conference*, 2021, pp. 52–63.

[26] S. Motepalli and H.-A. Jacobsen, "Reward mechanism for blockchains using evolutionary game theory," in *2021 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, 2021.

[27] T. Libra, "An introduction to libra," From the Libra Association Members, Tech. Rep., 2019, retrieved 10 Dec. 2020. [Online]. Available: https://libra.org/en-us/whitepaper

[28] D. Williams, "Sybil-resistant Network Identities From Dedicated Hardware," [Online; accessed 19-Oct-2020]. [Online]. Available: https://docs.google.com/document/d/1eRTAe3szuIoZEloHvRMtZlrU7t2un4UVQ8LarpU3LNk/

[29] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," in *Annual international cryptology conference*. Springer, 2018, pp. 757–788.

[30] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," Ph.D. dissertation, 2016.

[31] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 51–68.

[32] R. L. Rivest, A. Shamir, and D. A. Wagner, "Time-lock puzzles and timed-release crypto," 1996.

[33] D. Boneh, B. Bünz, and B. Fisch, "A survey of two verifiable delay functions." *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 712, 2018.

[34] B. Wesolowski, "Efficient verifiable delay functions," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2019, pp. 379–407.

[35] K. Pietrzak, "Simple verifiable delay functions," in *10th innovations in theoretical computer science conference (itcs 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[36] V. Attias, L. Vigneri, and V. Dimitrov, "Implementation study of two verifiable delayfunctions." *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 332, 2020.

[37] S. Alqahtani and M. Demirbas, "Bottlenecks in blockchain consensus protocols," *arXiv preprint arXiv:2103.04234*, 2021.

[38] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, no. 1999, 1999, pp. 173–186.

[39] B. Y. Chan and E. Shi, "Streamlet: Textbook streamlined blockchains," in *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, 2020, pp. 1–11.

[40] S. Blackshear, E. Cheng, D. L. Dill, V. Gao, B. Maurer, T. Nowacki, A. Pott, S. Qadeer, D. R. Rain, S. Sezer *et al.*, "Move: A language with programmable resources," *Avalaible at: https://developers.libra.org/docs/move-paper (Consulted on April 1, 2020)*, 2019.

[41] M. Patrignani and S. Blackshear, "Robust safety for move," *arXiv preprint arXiv:2110.05043*, 2021.

[42] J. Truby, "Decarbonizing bitcoin: Law and policy choices for reducing the energy consumption of blockchain technologies and digital currencies," *Energy research & social science*, vol. 44, pp. 399–410, 2018.

[43] J. Sedlmeir, H. U. Buhl, G. Fridgen, and R. Keller, "The energy consumption of blockchain technology: beyond myth," *Business & Information Systems Engineering*, vol. 62, no. 6, pp. 599–608, 2020.

[44] L. Chen, L. Xu, N. Shah, Z. Gao, Y. Lu, and W. Shi, "On security analysis of proof-of-elapsed-time (poet)," in *International Symposium on Stabilization, Safety, and Security of Distributed Systems*. Springer, 2017, pp. 282–297.

[45] L. Srinivasan, Balaji S.and Lee, "Quantifying Decentralization," [Online; accessed 10-Dec-2021]. [Online]. Available: https://news.earn.com/quantifying-decentralization-e39db233c28e

[46] Q. Lin, C. Li, X. Zhao, and X. Chen, "Measuring decentralization in bitcoin and ethereum using multiple metrics and granularities," in *2021 IEEE 37th International Conference on Data Engineering Workshops (ICDEW)*. IEEE, 2021, pp. 80–87.