A Partition Resilient Overlay Network via Blockchain

Vijeth Tumkur Aradhya
National University of Singapore
varadhya@comp.nus.edu.sg

Seth Gilbert
National University of Singapore
seth.gilbert@comp.nus.edu.sg

Aquinas Hobor
National University of Singapore
hobor@comp.nus.edu.sg

***Abstract*—Blockchains use peer-to-peer networks to handle underlying inter-peer communication, and these networks currently do not have any provable guarantees for desirable properties such as small diameter or partition-resilience. This is not just a theoretical problem: Heilman et al. [1] exploited bugs in `bitcoind` to isolate a subset of peers in the Bitcoin network by poisoning their address tables. Cryptocurrency blockchains are safety critical systems, so we need principled algorithms to maintain their underlying overlay networks.**

Our key insight is that we can leverage the blockchain itself to share information among the peers, and thus simplify the overlay maintenance process. We provide communication-efficient protocols to maintain a dynamic hypercubic network for proof-of-work blockchains. Given that the peers have restricted computational resources, and $\rho < 0.5$ constant fraction of peers are Byzantine, we prove that the network (formed by the honest peers) remains connected for a polynomial number of rounds with high probability, even with polynomial variation in the size of the network. Moreover, these properties hold despite significant churn. A key contribution is a secure mechanism for joining the overlay that relies on the blockchain to help new peers to contact existing members of the overlay.

Furthermore, by examining how peers join the system, i.e., the "bootstrapping service", we give a lower bound showing that (within log factors) our overlay network tolerates the maximum churn rate possible. In fact, we can give a lower bound on churn for any fully distributed service that requires connectivity.

***Index Terms*—dynamic networks, bootstrapping, Byzantine failures, blockchain**

I. Introduction

Peer-to-peer (P2P) overlay networks are widely used to provide a low-level communication infrastructure for high-level services. For example, most cryptocurrencies like Bitcoin [2] rely on the network to provide timely and efficient communication among the participants. Each peer only communicates with a small number of other peers from its local view to disseminate/route messages. By ensuring redundancy of data across peers, these systems are also robust against massive failures. P2P systems are also highly dynamic in nature and subject to high churn, wherein a large number of peers can (concurrently) join and leave the network at any given instant of time. This poses a unique challenge for peers to continually refresh information about the network in an efficient way. For example, each peer must keep updating its local view so that obsolete peers are replaced by new, fresh peers which may have joined a different part of the network. Moreover, it is notoriously difficult to maintain a dynamic overlay network when a constant fraction of the peers are malicious. The malicious peers can stay for an extended period of time, create many identities at once, and stealthily harm the network.

To maintain a global property of the underlying overlay network (such as connectivity, topology, etc), peers need to continuously distribute shared information for coordination. This coordination problem is exactly what blockchains are designed to solve. The key innovation of this paper is to utilize the data on the blockchain to coordinate the maintenance of the overlay network. But the blockchain itself relies on the overlay network to provide a global consensus amongst the peers. Hence, the main challenge of this paper is to carefully handle this circular dependency. Furthermore, we identify and (provably) handle three main issues of current blockchain networks: (1) secure and bandwidth-efficient bootstrapping of new peers, (2) maintaining low diameter and degree for the underlying overlay graph, and (3) remaining connected for a sufficiently long period of time.

In Bitcoin and many other blockchain networks, a major problem is a partitioning (or eclipse) attack, in which attackers isolate a subset of peers from the rest of the network. By taking over both the incoming and outgoing connections of those peers, the adversary is able to control the information flow between them and the rest of the network. Recent work [1], [3] has exposed network-level vulnerabilities in Bitcoin and Ethereum that allowed isolating victim nodes with low computational resources. In the context of cryptocurrency blockchains, this attack can be used as a primitive for other attacks such as double spending, reducing *effective* honest resources (by forcing a significant fraction of peers to mine on top of an *orphan* block), and selfish mining [4], [5].

Eclipse attacks have been a prominent issue in the design of overlay networks [6], [7]. Over the past couple of decades, there has been exciting theoretical work [8]–[13] in the context of robust distributed hash tables (DHTs). The general idea to ensure fault-tolerance is to maintain clusters of size logarithmic in network size, and use majority voting for routing and storing data items. Most notably, Awerbuch and Scheideler [12] introduced a suitable churn model to capture join-leave attacks by Byzantine peers to overwhelm clusters. They showed that certain structured overlay networks with good routing properties can be maintained with polylogarithmic complexity withstanding polynomial number of join-leave operations with high probability [12], [13]. In the same model

but allowing polynomial variation in network size over time, Guerraoui et al. [14] showed that with high probability, clusters of honest majority can be maintained by using random walks with expander graphs with small degrees. In general, however, these algorithms assume that peers joining the network have some type of global view of (or access to) the network, able (for example) to contact honest peers [15]–[17].

How new peers join the network is a critical aspect of the system. To facilitate churn, new peers need to contact some existing (honest) peers. Since P2P networks can be decentralized, it is often not clear as to who should be responsible for helping new peers join the network. Currently, cryptocurrency blockchains offer two main options[1] for bootstrapping peers: (1) via DNS seeding and (2) hard-coded IP addresses (in the shipped software) [17]. Hard-coded seeds are usually a fallback mechanism if DNS seeding mechanism fails. This is because most of the peers are subjected to churn, and the hard-coded IP addresses may become obsolete after a period of time. DNS seeding itself requires trusting a few sources to respond with random IP addresses that are within the network[2] [16]. Moreover, if there is a bandwidth constraint on each peer, and there are only a handful of DNS servers responsible for bootstrapping, then the system cannot withstand high churn.

To address these issues, we design protocols to maintain a hypercubic overlay network in an underlying synchronous network, wherein each peer sends or receives $O(\text{polylog}(N))$ bits in any round. A constant fraction $\rho < 0.5$ of the peers can be Byzantine. All the peers run a proof-of-work blockchain protocol to maintain a distributed ledger (which we treat mostly as a blackbox in this paper). The average number of rounds for a block to be generated by the entire network is β.

The network is subject to churn, where the honest peers have a "half-life" of α rounds, i.e., α is the smallest halving or doubling time: halving time is the number of rounds for half the (honest) peers to leave or fail; doubling time is the number of rounds for n new (honest) peers to join a system of size n, doubling the size (when no peers leave). The half-life restricts the honest peers; Byzantine peers can stay in the network for an indefinite period of time.

A key assumption is that the peers can utilize the blockchain as a public bulletin board [18] whose contents can be read by anyone. The peers (within the network) have write access to the board, i.e., each peer can add a small amount of information to the block that it is mining. The latest (confirmed) copy of the blockchain is available to the public at all times, and this provides an entry point to the service. Blockchain explorers [19]–[21] satisfy this requirement to some extent; there might be multiple copies available but the *confirmed* chain is more or less the same. (There might be some contention about the last few blocks among those explorers.)

To ensure fault-tolerance, each peer simulates $O(\log N)$ (virtual) *nodes* in the network. Nodes are grouped into committees of small size. These committees in turn form the vertices of a hypercube. Moreover, as soon as a peer mines a block, it is "elected" to help in bootstrapping new peers for a brief period of time. It does so by having selected one of its nodes to be a *directory* node whose network address information is on that newly mined block. The directory nodes are responsible for helping new peers to safely join the network. A key advantage of our protocol is that the nodes in the same directory are only loosely synchronized: they do not need to run consensus or other sorts of agreement protocols.

In summary, our main contributions are stated below (that hold with high probability):

1) We design protocols to maintain a dynamic hypercubic network where half-life $\alpha = \Theta(\sqrt{\beta N})$ and the graph formed by the honest peers remains connected for polynomial number of rounds, even with polynomial variation in the size of the network over time. Each peer has low degree, and expends only $O(\text{polylog}(N))$ bits per round. Moreover, we ensure that a new peer can contact an honest peer within the network.
2) Barring $\log N$ factors, we provide a matching lower bound for $\alpha = \widetilde{\Omega}(\sqrt{\beta N})$ (i.e., the maximum dynamic churn a system can have), if the distributed service requires connectivity and is implemented using any public bulletin board.

II. Other related work

There has been a large body of work on the design of robust and efficient overlay networks in various models of churn and failure. We mainly focus here on the Byzantine setting. We split them into two categories based on the differences in the churn model. We also survey practical partitioning attacks and network algorithms pertaining to blockchain P2P networks.

Bortnikov et al. [22] showed that using a gossip-based protocol, each (honest) peer can generate samples that converge to (uniformly) random samples of peers, which helps in ensuring that honest peers have at least one other honest peer in their local view. Their analysis of convergence holds only when the churn ceases. Fiat and Saia [8], [23] proposed a content addressable network that is robust against an adversary that can remove up to a constant fraction of peers. One drawback is that a constant fraction of honest peers may not be able to access some data items; another drawback is that the network size is fixed. Many DHTs have been designed assuming random Byzantine faults where each peer (independently) is assumed to have a fixed probability of being Byzantine. Naor and Wieder [24] used their continuous-discrete approach [10] to design a simple DHT where each data item is stored by logarithmic number of peers (which has a honest majority) and each query is routed to the destination through a majority vote at each hop. Hildrum and Kubiatowicz [9] took an iterative routing approach to design fault-tolerant routing algorithms when the network forms a restricted growth metric space and when the network distance can be securely measured. Fireflies [25]–[27] provides an overlay maintenance protocol to support full membership information. However, it is unclear how it

[1]There exist other bootstrapping mechanisms such as IRC channels, Tor, etc, which suffer the same problems.

[2]For example, Bitcoin lists a few DNS seed options and the people that are responsible for those hostnames.

performs in highly dynamic, large-scale and decentralized blockchains.

Awerbuch and Scheideler [12] identified a new class of attacks called join-leave attacks where Byzantine peers try populate a specific region of the overlay topology by repeatedly leaving and joining the network. They show that a dynamic distributed name service can be efficiently maintained for a polynomial number of join-leave operations with high probability, assuming that at most a $O(1/\log N)$ fraction of the nodes can be Byzantine, where N is the number of honest nodes. The churn rate for honest and adversarial nodes is also assumed to be different. The main idea is to enforce a limited lifetime for each node (controlled by a peer). The rest of the algorithms assume a constant fraction of the peers to be adversarial. Fiat et al. [11] modify Chord [28] to ensure robustness for a linear number of adversarial join-leave operations. Awerbuch and Scheideler [13] introduced the cuckoo rule to handle polynomial number of join-leave operations with high probability, without enforcing limited lifetime to nodes. Guerraoui et al. [14] maintain clusters of logarithmic size with honest majority, over a polynomial number of join-leave operations, in which the clusters form a small degree expander graph. This is guaranteed even with polynomial variation in the network size, unlike the previous works where the network size changes by at most a constant factor.

Heilman et al. [1] first demonstrated eclipse attacks on the Bitcoin network using low computational resources. Similarly, Marcus et al. [3] exploited a few vulnerabilities in Ethereum's P2P network to present eclipse attacks using just two machines (each with a single IP address). There have also been practical attacks that consider strong network infrastructure adversaries [29], [30]. In light of these issues, there have been new network design proposals for blockchains. Kadcast [31] further builds on Kademlia [31] and proposes a structured broadcast protocol for disseminating blocks with at most a logarithmic number of hops and constant overhead in congestion. It is unclear how this protocol performs with respect to continuous node churn and change in network size. In Perigree [32], a peer retains the "best" subset of neighbours after regular intervals, and also continuously connects to a small set of random peers to explore potentially better-connected peers. But Perigree may actually be more prone to eclipse attacks because the adversary can easily monopolize victim peer's connections by providing well-connected peers.

III. Model

Entities. A peer is the real-world entity that participates in the blockchain network. There are two types of peers: (1) honest peers that follow the specified protocols, and (2) Byzantine peers that may deviate from the protocols in an arbitrary way. The network size refers to the total number of peers within the network; the maximum network size is denoted by N. We assume that the network size can polynomially vary over time; the number of peers at any round r, $N_r \in [N^{1/y}, N]$ for some constant $y > 1$.

Communication. Each peer u maintains a set of *neighbouring* peers $\mathcal{N}_u$ that it can send messages to; intuitively, and peer can communicate with any peer whose "address" it knows. The system proceeds in rounds; in each round, in addition to the local computation, a message that is sent at the beginning of a round by peer u is assumed to reach its neighbours $\mathcal{N}_u$ by the end of that round. There is no global clock (to record the global round number) but the local clocks of peers run at approximately same speed. Our protocols are designed to have a small bandwidth cost; each peer sends or receives only $O(\text{polylog}(N))$ bits in each round for overlay network maintenance. Let Δ_{br} be the maximum number of rounds (that is determined by our overlay network) required for broadcasting a message to the entire network.

Computational restriction. The peers are associated with a *computational* or *hash power* constraint, i.e., each peer owns one unit of hash power which allows the peer to query a hash function (modelled as a random oracle) $q > 1$ times in a round [33], [34]. If some entity has more hash power, then it can be viewed as a coalition of peers. Given an input of any length, the hash function is assumed to provide a random output of (fixed) length $\kappa = \Omega(\log N)$.

Blockchain. Our primary focus is on proof-of-work P2P networks to support blockchains, wherein all the peers maintain a (consistent) distributed ledger. Specifically, they maintain a sequence of *blocks* called a *chain*; each block has a link to the previous block in the chain. The peers *mine* or repeatedly query the hash function to generate blocks in the chain. They follow a proof-of-work based blockchain protocol that is secure when $\rho < 0.5$ fraction of the peers are Byzantine, where ρ is some constant. The blockchain implementation will rely on the overlay network protocols presented in this paper to broadcast information to other peers. Furthermore, the blockchain satisfies the following four properties [34], [35].

Consistency. There exists a notion of *confirmed*[3] chain C_u^r for any honest peer u in round r such that C_u^r is always a prefix of $C_u^{r'}$ for any round $r' \geq r$. If $|C_u^r|$ is the number of blocks in the confirmed chain of honest peer u at round r, then by round $r + \Delta_{br}$, every honest peer's confirmed chain's length is at least $|C_u^r|$. For any two honest peers u and v at rounds r and r' respectively, if $|C_u^r| \leq |C_v^{r'}|$, then C_u^r is a prefix of $C_u^{r'}$.

Growth. The number of new blocks that get confirmed in the blockchain is proportional to the number of rounds elapsed. More formally, for $T = \Omega(S/\beta)$, $S \gg \beta$ where β is the average number of rounds for 1 block to be confirmed, let $T = |C_u^{r'}| - |C_u^r|$ for any honest peer u and rounds r and $r' \geq r$, then $T\beta/\mu_b \leq r' - r \leq \mu_b T\beta$ for a small constant $\mu_b \geq 1$, except with probability $2^{-\kappa} + 2^{-T}$.

Fairness. Except with probability $2^{-\kappa}$, any set of honest peers controlling a ϕ fraction of hash power is guaranteed to get at least $(1 - \delta)\phi$ fraction of the blocks in any $\Omega(\kappa/\delta)$ length segment of the chain.

[3]For example, in many proof-of-work blockchains that rely on the "longest chain" rule, blocks are confirmed when there are enough blocks added to the chain after them.

Public availability. Every peer that joins the system has access to a (reasonably) up-to-date copy of the blockchain. (In practice, this typically means there exist a set $\mathcal{P}$ of publicly available blockchains, of which at least some are honest.)

We specify the interface with which any peer u interacts with the blockchain protocol.

1) `GET_CONF_CHAIN`. Returns the confirmed chain C_u^r of peer u at round r.
2) `ADD_TO_BLOCK`. Adds $O(\log N)$ bits of information to the block that is being mined by peer u.

Adversary. The adversary's main goal is to disconnect the network formed by the honest peers, or otherwise prevent the overlay form working as guaranteed. We consider Byzantine peers that can collude and arbitrarily deviate from the specified protocols as the adversary [10], [13], [14]. In any round, the number of Byzantine peers is most ρ fraction of the total network size. We also assume that the adversary cannot break any cryptographic primitive. Moreover, the adversary cannot modify or delete the messages sent by honest peers.

Churn. Following Awerbuch and Scheideler [12], [13], we consider a mix of oblivious and adaptive adversary to model churn. The adversary has to specify a sequence of join or leave operations for the honest peers in advance. And at any point in time, the adversary may either choose to rejoin one of its own peers in the network, or choose the next operation in that sequence. We ensure that the adversary (and honest peers) does not generate too many nodes per round through proof-of-work puzzles. This automatically puts a constraint on the number of rejoins that an adversary can perform per round. This particular model is useful for analyzing join-leave attacks in which the adversary aims to populate a specific region of the network.

Churn rate. We adopt Liben-Nowell et al. [36] half-life measure to model the churn rate for honest peers. Formally, at any given round r, the halving time is the number of rounds taken for half the number of honest peers (which were alive at round r) to leave the network; similarly, the doubling time is the number of rounds required for the number of honest peers to double. There is no distinction made between gracefully leaving the network and a benign (crash) failure of a honest peer (peers do not need to follow on any protocol for leaving the network). An *epoch* is defined as the smallest halving time or doubling time over all rounds in the execution; it is denoted by α. Furthermore, we assume that the epoch is much greater than the average number of rounds to generate a block; in other words, $\alpha \gg \beta$.

Honest peer failure. By the churn rate assumption, at most half of the honest peers that are alive at round r can leave (or fail) by the end of $r + \alpha$ rounds. The peers are effectively anonymous (from the perspective of the adversary), generating random ids that are used to communicate during the protocol. Since the adversary has to obliviously specify a join/leave sequence before the execution begins (and without knowing which peers will play which role in the protocol), we can think of the peers as being chosen to fail uniformly at random in each epoch (at least with respect to a given random id generated during the protocol). That is, each honest peer (independently) fails with a probability $p_f \leq 0.5$ in each epoch where p_f is the fraction of honest nodes that fail during the epoch.

Change of network size. We assume that the network size can change by at most σ_d factor in any epoch, where $1 < \sigma_d \leq 2$ is some constant.

IV. Stable Network Size

In this section, we assume that the total number of peers is fixed and equal to $\Theta(N)$. In Section V, this assumption is relaxed, where we allow a polynomial variation in the network size over time. (For simplicity, we avoid using floor/ceiling repeatedly unless necessary.) We now describe the basic overlay maintenance protocols. Due to lack of space, we defer the pseudocodes for protocols to the full version [37].

Nodes. Each peer generates and controls a small number of (virtual) *nodes* that it uses to populate the overlay network. Each peer participates (broadcasts and relays messages) in the network through its nodes. All peers are required to perform proof-of-work to generate nodes. Each node has a *lifetime* after which it will be considered invalid (or removed). There are two *roles* of a node: a (1) directory node, and a (2) non-directory node. Every node is initially a non-directory node, but some nodes become directory nodes as well, playing both roles.

Committees. Our protocols maintain a structured network of *committees*, specifically, a hypercube whose vertices correspond to committees. In total, there are $\mathcal{C} = N/\log N$ committees. And each committee has $\Theta(\log N)$ nodes. As a peer may be controlling multiple nodes, it may be present in multiple committees.

Directory node. A peer that successfully adds a block to the blockchain promotes one of its nodes to a directory node. While trying to mine for blocks, each peer adds a node ID and network address into the prospective block. The directory nodes are responsible for helping incoming new nodes to join the network. They do so by providing the network addresses of the required committee members.

A. Directories

Directory nodes are critical for maintaining information about the overlay members. The members of the directory are identified by information stored in the blockchain with one node per block. A directory comprises $\mathcal{B}$ consecutive "buckets", and each bucket consists of (consecutive) $\lambda_d \log^2 N$ blocks, where λ_d is a suitable constant. Each peer that creates a block added to the confirmed chain becomes part of one of the buckets in the directory (via one of its nodes). The total number of directory nodes is equal to $\mathcal{K} = \mathcal{B}\lambda_d \log^2 N$.

Functions. Each bucket is *responsible* for a set of committees, i.e., all the directory nodes in that bucket are supposed to help new nodes join a specific set of committees. There are two main functions of a directory node.

1) A directory node *stores* information about nodes belonging to a set of committees. In particular, if a new node joins one of those committees, then it stores the

new node's *entry information*[4] (that includes node ID, network address, etc) in the appropriate set.

2) A directory node also *sends* information about the committees that the directory node is responsible for. In particular, the directory node sends entry information about all the nodes that belong to the committee that the new node wants to join, or one of its neighbouring committees.

Committee-Directory mapping. There exists a predetermined mapping $\mathcal{M} : [\mathcal{C}] \to [\mathcal{B}]$ from committees to buckets. The mapping ensures that each bucket is responsible for (almost) same number of committees. The sets of committees that any two buckets are responsible for, are disjoint. This mapping is necessary for new nodes joining the network, to know which directory nodes to contact, to get information about relevant committees. It is also required for a peer to figure out the set of committees that its directory node is responsible for.

Active directory. The *active* directory is a set of directories that forms the bootstrapping service. It is useful to think of an active directory as a sequence of buckets which "wraps around" if the bucket number is more than $\mathcal{B}$. This means that there are multiple buckets that responsible for a given set of committees. Let the number of blocks and number of buckets in an active directory be denoted as $\mathcal{K}_{act}$ and $\mathcal{B}_{act}$ respectively.

Phases of bucket. The buckets can belong to one of the following four phases; the directory nodes are also classified into those four phases in the same way as the buckets. We describe these phases, transitioning from one phase to another, and also a directory node's behaviour at each phase. We refer to the `DIR` protocol pseudocode in [37] for more details.

1) *Infant.* A bucket in which at least one block (out of $\lambda_d \log^2 N$) is confirmed, but not all the $\lambda_d \log^2 N$ blocks are confirmed yet. During this phase, the nodes do not store the incoming new nodes' entry information. They do not respond to the incoming new nodes about any committees.
2) *Infant to middle-aged.* If all $\lambda_d \log^2 N$ blocks of the bucket are confirmed in the blockchain, then the bucket transitions to middle-aged phase.
3) *Middle-aged.* A bucket in which all the $\lambda_d \log^2 N$ blocks are confirmed, and these nodes store incoming new node entry information, and reply back to them about the relevant committees' information that they know about.
4) *Middle-aged to veteran.* All the directory nodes wait for Δ_{br} rounds after the next bucket that is responsible for the *same* set of committees is confirmed, before switching to veteran phase. In other words, as soon as the bucket is not one of the most recent (confirmed) $\mathcal{B}$ buckets, then the bucket transitions to veteran phase after a delay of Δ_{br} rounds[5].
5) *Veteran.* A bucket in which all the $\lambda_d \log^2 N$ directory blocks are confirmed; these nodes do not store new nodes' entry information, but they do reply about the relevant committees' information known to them.
6) *Veteran to dead.* As soon as the bucket is not one of the most recent (confirmed) $\mathcal{B}_{act}$ middle-aged or veteran buckets, then the bucket transitions to dead phase after waiting for Δ_{br} rounds.
7) *Dead.* In this phase, all the directory nodes in that bucket neither store any new node information, nor reply to new nodes about any committee information.

B. Joining the network

All the nodes have a limited lifetime after which the nodes would be considered "invalid". Due to this limited lifetime, all peers would need to regularly generate and incorporate new nodes into the network. But this could enable the adversary to generate and control too many nodes. To handle this issue, our system utilizes proof-of-work as Sybil resistance to limit the number of nodes controlled by any peer in any round, requiring peers to continuously mine for nodes, in addition to blocks. The peers can simultaneously mine for both nodes and blocks without spending extra computational resource by using the 2-for-1 PoW technique [33], [35], though the details of the proof-of-work scheme are beyond the scope of this paper.

Proof for joining. Let N_c be the nonce, $\hat{\mathrm{B}}_l$ be the hash of the latest confirmed block, and P_{key} be a self-generated public key of the peer. Then, the peer evaluates,

$$P_{join} = \mathbf{H}(\hat{\mathrm{B}}_l \parallel N_{id} \parallel N_c),$$

to join the network through a new node. If $P_{join} < T_{join}$, where T_{join} is the *mining target* for joining the network, then the node is considered to be a "valid" node, which means that the peer would be able to communicate with the bootstrapping service to register that node, and (re)join the network. The first $\log N$ bits of P_{join} represent the (random) committee number to which this new node would belong to. Let p_{st} be the *difficulty threshold* for node mining. It is the probability that a single hash query is successful in generating a valid node. T_{join} is set such that, in each epoch, the expected number of valid nodes that can be generated is equal to $p_{\mathrm{st}} q \alpha N = \lambda_n N$.

Entering the network. We now briefly describe the steps taken by a peer to generate and join a new node into the network. See pseudocode of `JOIN` protocol [37] for more details.

1) The peer expends its computational resources to first generate a valid node. Recall that the node is valid only if $P_{join} < T_{join}$.
2) Once a valid node is generated, the peer signs its network address concatenated with P_{join} using its secret key S_{key}. A node's *entry information* constitutes this signature, peer's public key P_{key}, the nonce N_c and the

[4]See Step 2 of `JOIN` protocol in Section IV-B for the exact definition. The word "entry" may be dropped when the context is clear. A committee's information refers to the set of all its nodes' entry information.

[5]The delay of Δ_{br} rounds is to ensure that all peers' confirmed chains reach the same height before making those transitions.

block number of the block that was used while mining for that node. Then, it sends a JOINING message with the entry information to only the directory nodes that are responsible for serving either the committee that the peer is going to join, or its neighbouring committees.

3) Those directory nodes verify the proof and respond to the peer with COMM_INFO message that has the entry information of the existing nodes of that committee and the neighbouring committees. The middle-aged directory nodes that responded with COMM_INFO message store the entry information of this new node.
4) The new node joins the network by providing the (same) JOINING message to all the nodes in that committee and its neighbouring committees.

In Step 3, the key requirement is that there is at least one honest directory node to respond. There may be some malicious directory nodes that can reply back with a wrong COMM_INFO message. By doing some basic checks (including verifying the entry information of nodes that were sent by directory nodes), the new node can thwart such wrong messages. If there always exists one honest directory node in each bucket of the active directory, and if the (new) peer takes a union of all the directory responses, the new node can get the information of all honest nodes in the required committees.

C. Lifetime of nodes

To conduct an eclipse attack on some peer, the adversary would need to populate its neighbour list; in other words, control all the peer's connections. Having a limited lifetime for a node prevents the adversary to populate a specific region in the network. If a node's lifetime is too low, then this would increase the number of join requests per round. On the other hand, if the node's lifetime is too high, then this may enable the adversary to conduct eclipse attacks by making use of churn on honest peers. It is worth noting that the churn rate for honest peers (and average block generation rate) is a determining factor for setting the lifetime of a node. In Theorem 5, we show that the proper lifetime of a node is $O(\alpha \log N/\beta)$ blocks.

Lifetime of non-directory node. Peers attach the hash of the most recent confirmed block B_l (that they are aware of) as part of the input to the hash function when they are mining a new node. As there is no global clock, the proof for joining conveniently records the new node's "entry time" (in terms of block number). This helps the other peers to keep track of a node's "age" as they would have received its entry information in their first interaction with the node. The lifetime of a non-directory node, specified in terms of number of blocks, is $T_l = \lambda_l \alpha/\beta$ blocks, where λ_l is a suitable constant. The node u that had joined at block b_l would be considered invalid after block $b_l + T_l$ is confirmed, where b_l is the block number of block B_l, at which point the peer stops controlling that node u, and all the other peers that had node u as its neighbour remove u from their nodes' neighbour list.

Lifetime of directory node. If a node is promoted to a directory node, then it obtains another life (which is separate from the non-directory life). When a node becomes a directory node, it continues to perform the functions of a non-directory node as long as the non-directory role is considered to be valid. The directory node is considered to be alive for T_{dl} blocks since the block in which it is embedded in. We set T_{dl} to be $\lambda_{dl}\alpha/\beta$ blocks, where $\lambda_l < \lambda_{dl}$ is some constant.

D. Setting parameters

Constraints on α and $\mathcal{B}$. A natural constraint on α and $\mathcal{B}$ arises due to the entry of new nodes. Let $\mathcal{U} = O(\mathrm{polylog}(N))$ be the highest number of join requests that can be handled by one directory node per round. By our model of churn rate, the system must be able to allow (at least) $\lambda_n N/2$ nodes to enter the network in any α (consecutive) rounds. A new node contacts all the directories within the active directory while joining the network (as it contacts both the middle-aged and veteran buckets). Therefore, for any directory,

$$\alpha \mathcal{B} \mathcal{U} \geq (\lambda_n N \log N)/2,$$

where the LHS of the inequality is the total number of join requests that handled in one epoch, and the RHS represents the minimum number of join requests that can be generated in one epoch. There is an extra $\log N$ factor because a new node can contact $\log N$ different buckets[6] in the worst case (depending on the committee-directory mapping).

Let us calculate the value of $\mathcal{K}_{act}$. This value is determined by how long a directory node needs to stay in each of its phases (from infant to veteran). Firstly, if a directory node is the first node to be part of a bucket, then it needs to be alive for $\lambda_d \log^2 N$ blocks (size of a bucket). Secondly, recall that a directory node stays in the middle-aged phase until the next bucket that is responsible for the same set of committees is formed, and this takes $\mathcal{K}$ blocks. Finally, recall that a directory node stops storing node entry information when it transitions to veteran phase. Thus, a directory node needs to be in veteran phase only until some (non-directory) node in one of the committees that it is responsible for, is still alive. And this takes at most T_l blocks. Therefore, the active directory has $\mathcal{K}_{act} = (1+\mathcal{B})\lambda_d \log^2 N + \lambda_l \alpha/\beta$ number of blocks. And the number of buckets in the active directory is $\mathcal{B}_{act} = \frac{\mathcal{K}_{act}}{\lambda_d \log^2 N}$. The lifetime of directory node T_{dl} may be slightly more than $\mathcal{K}_{act}$ because there is a delay of Δ_{br} rounds in the transitions. And this automatically gives rise to the following constraint on α and $\mathcal{B}$,

$$(1+\mathcal{B})\lambda_d \log^2 N + \lambda_l)\alpha/\beta \leq \lambda_{dl}\alpha/\beta.$$

Setting α and $\mathcal{B}$. Due to the aforementioned constraints, we ensure that $\alpha = \Theta(\sqrt{\beta N})$ by setting $\mathcal{U} = O(\mathrm{polylog}(N))$ appropriately. And $\mathcal{B}$ is set such that $\mathcal{K}_{act}$ is slightly less than T_{dl} to account for Δ_{br} delay.

E. Bootstrapping service

The most recent $\mathcal{B}_{act}$ consecutive (confirmed) buckets in the blockchain, which are either in middle-aged or veteran phase,

[6]This can go up to $\log^2 N$ different buckets in Section V.

form the bootstrapping service. The new nodes must figure out the sequence of buckets in the active directory (using the set $\mathcal{P}$ and committee-directory mapping), and contact the relevant buckets to get the required committees' entry information for joining the network.

F. Bootstrapping network

We make an assumption that at time zero, the network has formed a hypercube structure of committees, with $\Theta(\log N)$ honest peers in each committee, and $\Theta(\log^2 N)$ honest nodes in each bucket of the first (active) directory, and each peer controlling at most $O(\log N)$ nodes. If N_0 is the number of peers in the beginning of the execution, then the number of committees $\mathcal{C}$, is set such that $N_0 = \mathcal{C} \log N$. In each committee, the network formed by honest peers is connected, and each committee has $\Omega(\log N)$ (different) honest links to its neighbouring committees. Moreover, the difficulty threshold for node mining is appropriately set (according to the initial network size). At this point, the network allows dynamic participation and satisfies the required properties.

V. Dynamic Network Size

As seen in Bitcoin [2] and other proof-of-work blockchains, the total hash power in the system can substantially change over time. For simplicity, we stick to the model where each peer can query the hash function a fixed number of times in a round, therefore, change in hash power actually refers to change in the number of peers. It is crucial that our overlay maintenance protocols are robust to change in network size over time.

There are two main problems that arise when the network size is allowed to significantly vary over time. Firstly, the mining targets for node generation (and block generation) should be regularly reset according to the network size. This is required to maintain a small number of nodes in any committee. Secondly, the dimension of the hypercube must also be changed accordingly. This is because if the network size keeps decreasing and the number of committees remain the same, then each peer would need to simulate too many nodes at any time, for the system to maintain $\Theta(\log N)$ in every committee[7]. And if the network size keeps increasing and the number of committees remain the same, then some peers may not be able to participate in the network all the time because they may take too long to generate nodes.

Before diving into the details on how we handle the two aforementioned issues, we provide a high-level intuition of our approach. The main idea is to efficiently estimate the network size in every epoch. If all the peers can get a good estimate, then they can use the blockchain to agree on that estimate, thereby change the mining target and the dimension, if required.

Finally, how do we reset the mining target for node generation? The expected number of nodes that can be generated in any epoch e is kept as close as possible to the quantity $\mathcal{J}_e = \lambda_n \mu_b \mathcal{C}_e \log N$, where $\mathcal{C}_e$ is the number of committees in epoch e. If this is done, then in any epoch, the expected number of nodes that get allocated to any committee is $\Theta(\log N)$ (similar to Section IV).

[7]After dimension decrease, each committee may have up to $O(\log^2 N)$ nodes for a brief period of time.

A. Time in terms of b-epochs

In the dynamic setting, an epoch is divided into two "phases" which aids in describing the protocols for network size estimation and dimension change. Phase 1 is called the *estimation* phase where the existing peers calculate an estimate of the total number of nodes that joined in that phase to get a good estimate of the network size. There are α_1 rounds in phase 1. Phase 2 is called the *agreement* phase where the peers reach an agreement (via blockchain) on the new difficulty threshold and the decision to change dimension. There are α_2 rounds in phase 2. These two phases make up the epoch, $\alpha = \alpha_1 + \alpha_2$. The length of phase 2 is actually much smaller than length of phase 1 (by design, see Section V-C), i.e., $\alpha_2 \ll \alpha_1$; in particular, $\alpha_1 = \Theta(\alpha)$.

For the peers to be able to estimate the network size in the first phase of an epoch, they must be able to figure out when the epoch starts, and when the first phase of that epoch ends (without any global clock). As the peers are not aware of the (global) round number, they rely on the most recent confirmed block number to establish: (1) the start of an epoch, and (2) end of the first phase of an epoch. The idea is to fix multiples of appropriate block number to mark the start and the end of phase 1 of an epoch. Let us say that $\frac{\alpha}{\mu_b \beta}$ consecutive blocks is called a *b-epoch*. And analogous to an epoch, the length of phase 1 of a b-epoch is $\frac{\alpha_1}{\mu_b \beta}$ consecutive blocks. Thus, using this information and the most recent confirmed block number, the peers can run the protocols at (approximately) same time.

B. Network size estimation

The peers require a good estimate of the network size for changing the dimension of the hypercube and difficulty threshold for node mining. This is done in phase 1 of every b-epoch. We give a brief description of the `NET_SIZE_EST` protocol [37] run by each node.

1) Each node keeps track of the set of (new) nodes that join its committee from start of a b-epoch e to end of phase 1 of b-epoch e.
2) A random committee is picked at the end of phase 1 of b-epoch e. Let b_e^1 be the confirmed block that marks the end of phase 1 of b-epoch e. Let $h_e = \mathbf{H}(b_e^1)$ be the hash of the last (confirmed) block in phase 1. Let the first $\log \mathcal{C}_e$ bits of h_e determine this (random) committee.
3) All the nodes in this committee broadcast the entry information of nodes within their committee, that joined the committee in phase 1.
4) The peers in the other committees take the union of the responses to obtain the (same) list of nodes of that committee which joined in phase 1. Let H_e be the total number of those nodes.

5) An estimate of total number of new nodes that joined in phase 1 is $G'_e = H_e\mathcal{C}_e$. Using this estimate, the nodes calculate the network size estimate $M'_e = \mu_b G'_e/(p_e\alpha_1)$.

C. Resetting parameters

Once a peer gets a good estimate of the network size after phase 1, it can recalculate the difficulty threshold and the number of committees for the next b-epoch. All the peers utilize the blockchain to reach an agreement on these new values. In other words, whenever a peer mines a block in the next $\alpha_2/\mu_b\beta$ blocks, it adds these new values onto the block so that the incoming new nodes (and the directory nodes) know the updated threshold.

There is a delay of one b-epoch in switching to the next hypercube of a different dimension, i.e., if the peers decide to change the dimension during phase 2 of a b-epoch e, and they wait for one b-epoch, and then adopt the next hypercube in b-epoch $e+2$. The epoch before which the next hypercube is actually adopted, is called a *transformation* epoch. This decision is taken in b-epoch e assuming a worst case change of σ_d factor to the network size over the next two epochs; the decision depends on whether there is a possibility that their network size estimate in b-epoch $e+2$ is not in $[\mathcal{C}_e, \mathcal{C}_e \log^2 N]$.

More precisely, $trans\_epoch, p_{e+1}$ and $\mathcal{C}_{e+1}$ are added to every block that is mined during phase 2. Here, $trans\_epoch$ is a variable that determines whether the next b-epoch is a transformation epoch; if it is, then $trans\_epoch$ specifies dimension increase or decrease, and otherwise, $trans\_epoch$ specifies no change. See Algorithm 1 (PARAM_RESET protocol) for more details on how these values are calculated. These values are added to the next $\alpha_2/\mu_b\beta = \lambda_{lb}\log N$ blocks after the end of phase 1, where λ_{lb} is some constant. Due to the fairness property of the blockchain, the majority of those blocks belong to honest peers, which helps in reaching consensus on those values by the end of the epoch. Hence, we can set α_1 to be equal to $\alpha - \lambda_{lb}\mu_b\beta\log N$. This implies that $\alpha_1 = \Theta(\alpha) \gg \beta$.

D. Changing dimensions of hypercube

The system tolerates a factor of $O(\log N)$ change in the network size before changing the dimension, and this change does not happen too often in practice. Shifting from the old hypercube to the new one requires a time of one b-epoch. The system operates in the old hypercube in the transformation b-epoch. During the transition, the behaviour of existing nodes (that joined before the transformation b-epoch) is different for dimension increase and decrease.

Committee-Directory mapping. For each dimension change, the committee-directory mapping (see Section IV-A) needs to be appropriately changed such that no two buckets in the directory hold information about the same committee, and that each bucket holds (almost) the same amount of committee information. For each (possible) dimension, this mapping is predetermined. There may be almost no overlap (intersection of the sets of committees responsible by a bucket) between two (consecutive) committee-directory mappings. For this reason, the following algorithms rely on some buckets of the directory to serve the old hypercube and the new hypercube simultaneously for a brief period of time.

Algorithm 1 PARAM_RESET protocol

Require: Start this protocol at the end of phase 1 of b-epoch e. Let M'_e be the output obtained from NET_SIZE_EST protocol. δ_{err} is a small, fixed constant.

Ensure: Output whether b-epoch $e+1$ is a transformation epoch, and the values of $p_{e+1}, \mathcal{C}_{e+1}$.

$L'_e \leftarrow M'_e$.
$\mathcal{C}_{e+1} \leftarrow \mathcal{C}_e$.
$trans\_epoch \leftarrow$ "none".
if $\mu_b\sigma_d^2 L'_e > (1-\rho)(1-\delta_{err})\mathcal{C}_e \log^2 N$ **then**
  $\mathcal{C}_{e+1} \leftarrow \mathcal{C}_e \log N$.
else if $L'_e < \mu_b\sigma_d^2(1+\delta_{err})\mathcal{C}_e$ **then**
  $\mathcal{C}_{e+1} \leftarrow \mathcal{C}_e/\log N$.
else if $[\mu_b\sigma_d^3 L'_e > (1-\rho)(1-\delta_{err})\mathcal{C}_e \log^2 N]$ **then**
  $trans\_epoch \leftarrow$ "increase".
else if $[L'_e < \mu_b\sigma_d^3(1+\delta_{err})\mathcal{C}_e]$ **then**
  $trans\_epoch \leftarrow$ "decrease".
end if
$p_{e+1} = \frac{\lambda_n\mu_b\mathcal{C}_{e+1}\log N}{\alpha_1 L'_e}$.
Output $(trans\_epoch, p_{e+1}, \mathcal{C}_{e+1})$.

Directory. As soon as a transformation b-epoch e starts, the middle-aged buckets in the active directory start serving committees in two committee-directory mappings $\mathcal{M}_e$ and $\mathcal{M}_{e+1}$ for the current hypercube and the next hypercube which have $\mathcal{C}_e$ and $\mathcal{C}_{e+1}$ committees respectively. In other words, the directory nodes in those buckets, run DIR protocol for both hypercubes. The buckets that are responsible for storing and/or replying back entry information about committees in two hypercubes are said to be in a *split* state. All the buckets formed in this transformation b-epoch, including the initial middle-aged buckets, would be in the split state. And the buckets formed after the transformation b-epoch, only serve the next hypercube. The bandwidth cost of a directory node which is in split-state, is increased, but still remains within $\mathrm{polylog}(N)$ messages.

New nodes. The new nodes that join the network during the transformation b-epoch e, get a node in both the hypercubes by considering the first $\log\mathcal{C}_e$ and $\log\mathcal{C}_{e+1}$ bits of P_{join}. But they only (temporarily) operate in the old hypercube (with old mapping $\mathcal{M}_e$) in the transformation b-epoch e. Then, from the next b-epoch, this node (in the old hypercube) would be considered invalid, and they start operating using the node in the next hypercube. The new nodes contact the split state buckets and the buckets formed before them, for the entry information of nodes that joined before the transformation b-epoch using the old committee-directory mapping. And they contact the split state buckets and the buckets that are formed after them, for registering, and getting entry information about nodes that joined in or after the transformation b-epoch, using the new committee-directory mapping.

1) Dimension increase: The nodes that joined the network before the transformation epoch, will continue to be in the

same committees even after the protocol terminates though the committee number in the new mapping may be changed. These existing nodes will learn about the new nodes that join that committee (in both hypercubes) in the transformation b-epoch.

2) Dimension decrease: The nodes that joined before the transformation epoch, need to merge with neighbouring committees, to shift to the new hypercube by the end of the transformation b-epoch. Every $\Delta_{br} + 1$ rounds (from the start of the transformation b-epoch), each committee chooses a neighbouring committee (which is predetermined for each possible dimension) to merge with. Merging is done by exchanging entry information about each other's neighbouring committees. See DIM_DEC protocol [37] for more details. These (old) nodes need to wait for $\Delta_{br} + 1$ for each merge because all the (honest) peers may not start the transformation b-epoch at the same time. By the end of $O(\log \log N)$ merges, the algorithm ends wherein each committee has at most $O(\log^2 N)$ nodes.

VI. Analysis

We present proof sketches for the main theorem and three important supporting lemmas in this section. Due to lack of space, we defer the full analysis to the full version [37].

We split the time into b-epochs in the analysis. We prove that the network formed by honest peers remains connected assuming that the previous few b-epochs are "stable" and "bandwidth-adequate". A b-epoch e is said to be stable if (1) the expected number of new nodes that can be generated in that b-epoch is in $[C_1\lambda_n\mathcal{C}_e \log N, C_2\lambda_n\mathcal{C}_e \log N]$ for some fixed constants C_1, C_2, (2) the network size in any round in that b-epoch is in $[\mathcal{C}_e, \mathcal{C}_e \log^2 N]$. A b-epoch e is said to be bandwidth-adequate if each (honest) peer needs to send or receive $O(\text{polylog}(N))$ messages for network maintenance in any round. Finally, we show that b-epoch is stable with high probability if the network is connected and peers have a good network size estimate in the previous two b-epochs.

Theorem 1. *If $\alpha = \Theta(\sqrt{\beta N})$, then the network formed by honest peers is connected and each (honest) peer sends or receives $O(\text{polylog}(N))$ messages per round, for a polynomial number of rounds with high probability.*

The main idea is that if the network formed by honest peers is connected and the peers calculate $trans\_epoch, p_{e+1}$ and $\mathcal{C}_{e+1}$ in b-epochs $e-1$ and $e-2$, and if the b-epoch $e-1$ is bandwidth-adequate, then b-epoch e is a stable b-epoch. This is true because the dimension would be changed appropriately changed at epoch $e-1$. And also due to the bound on the maximum change of network size in any epoch, we can show that the properties of stable epoch hold for b-epoch e.

If the next b-epoch is a stable epoch, then we can show that the network remains connected and has other good properties with high probability. The following lemmas outline the main steps:

Lemma 2. *If the b-epochs $[e, e-1, \ldots, z+1, z]$ where $z = \max(1, e - \lceil \lambda_{dl}\mu_b \rceil)$, are bandwidth-adequate b-epochs, and if b-epoch $e-1$ is stable, then whp, the overlay network is partition-resilient in b-epoch e.*

Proof sketch. A key observation here is that if we can show that there is always at least one honest peer in each bucket of the active directory, and there is enough bandwidth to process all the join requests, then we can also show that the existing peers and new peers know about each other after executing JOIN protocol. This solves the connectivity problem for honest peers. It is given that the last $\lceil \lambda_{dl}\mu_b \rceil$ b-epochs are bandwidth adequate. By using fairness property, we show that there exists $\Omega(\log N)$ honest peers in each bucket. Then, by using Chernoff bounds combined with honest peer failure, we show that all the buckets so far and the ones formed in b-epoch e always at least one honest peer with high probability.

Finally, we show that in any stable b-epoch, each peer controls at most a constant number of new nodes (generated in that b-epoch) in any committee with high probability. Then, using the fact that the expected number of nodes generated in b-epoch $e-1$ is at least $C_1\lambda_n\mathcal{C}_e \log N$, we prove that each committee has $\Omega(\log N)$ honest peers in b-epoch e with high probability. This proves that an honest peer has $\Omega(\log N)$ (honest) links in each neighbouring committee. ■

Lemma 3. *If the b-epochs $[e, e-1, \ldots, z+1, z]$ where $z = \max(1, e - \lceil \lambda_l\mu_b \rceil)$, are stable b-epochs, then whp, b-epoch e is bandwidth-adequate.*

Proof sketch. Each node always has $O(\log N)$ neighbouring committees. In a stable b-epoch e, the expected number of nodes generated is at most $C_2\lambda_n\mathcal{C}_e \log N$. And the network size is in $[\mathcal{C}_e, \mathcal{C}_e \log^2 N]$. By a balls and bins argument, each committee has $O(\log N)$ nodes with high probability. Recall that the nodes that were generated before the last $\lceil \lambda_l\mu_b \rceil$ b-epochs are considered invalid. Due to dimension decrease algorithm, there may be $O(\log^2 N)$ nodes in a committee for a brief period of time.

The expected number of nodes that can be generated by one peer is $O(\log N)$ due to the properties of stable b-epoch. By using Chernoff bounds, and union bounding over all rounds and peers, we show that a peer controls $O(\log N)$ nodes in any round with high probability.

The tricky part is to show that bandwidth cost of a directory node is also low. First, consider the case where the network size is $\Omega(\alpha)$. Using the fairness property of the blockchain, we prove that each peer has at most $O(\log N)$ blocks in the active directory. Let D be the total number of valid join requests per round. In a stable epoch, the expected number of join requests per round is at most $C_2\lambda_n\mu_b^2\mathcal{C}_e \log N/\alpha$. And the network size is in $[\mathcal{C}_e, \mathcal{C}_e \log^2 N]$. If $\mathcal{C}_e = \Omega(\sqrt{\beta N})$, then by Chernoff bounds, $D = O(\mathcal{C}_e \log N/\alpha)$ with high probability. Since the nodes are randomly allocated to committees (by random oracle assumption), the load gets equally divided among the buckets (due to committee-directory mapping), by a balls and bins argument and the constraint in Section IV-D, each peer gets at most $O(\text{polylog}(N))$ join requests per round. Finally, for the case where network size is at most $c_1\alpha$ but greater than $N^{1/y}$

for some constant c_1, a peer may have too many blocks in the active directory. For a stable epoch e, $N_e \in [\mathcal{C}_e, \mathcal{C}_e \log^2 N]$, this would imply that $C_2 \lambda_n \mu_b^2 \mathcal{C}_e \log N/\alpha = O(\log N)$. By Chernoff bounds, with high probability, $D = O(\log N)$ join requests can be generated per round. And in this case, each (entire) directory in the active directory, gets $O(\text{polylog}(N))$ join requests (in total) with high probability. ■

Lemma 4. *If b-epoch e is stable and bandwidth-adequate, and the the network formed by honest peers is connected in b-epoch e, then whp, the (honest) peers can calculate the quantity M'_e such that $M'_e \in [(1-\delta_{err})(1-\rho)M_e^L/\mu_b, (1+\delta_{err})\mu_b M_e^H]$ where M_e^L and M_e^H are minimum and maximum network sizes in phase 1 of b-epoch e.*

Proof sketch. First, we show that all honest peers can get a good estimate of the number of nodes that joined in phase 1 of epoch e. As this is a stable b-epoch, we have bounds on the expected number of nodes that can join in phase 1 of epoch e. We get good concentration on the same by Chernoff bounds. These nodes (bins) get randomly allocated to the committees (balls). In `NET_SIZE_EST` protocol, a random committee is picked to estimate the number of nodes. This is akin to selecting a bin at random to estimate the number of balls. The peers calculate $G'_e = H_e \mathcal{C}_e$ where H_e is the total number of nodes which joined that committee. The peers then set the quantity M'_e to be $\mu_b G'_e/(p_e \alpha_1)$. And using the blockchain growth assumption and the bound on the maximum change in network size in any epoch, we can show that $M'_e \in [(1-\delta_{err})(1-\rho)M_e^L/\mu_b, (1+\delta_{err})\mu_b M_e^H]$ with high probability.

M'_e is used as the network size estimate to calculate $trans\_epoch, p_{e+1}$ and $\mathcal{C}_{e+1}$ as given in Algorithm 1. Then, using the fairness property, by the end of phase 2, all the peers agree on the new values with high probability. ■

VII. Lower bound for half-life

In the context of dynamic overlay networks, there is always a problem of bootstrapping a new peer into the network. How does a new peer contact an existing peer within the network? We consider implementing a bootstrapping or introductory service $\mathcal{I}$ that has two properties:

1) *Secure.* Responds with at least one honest peer that is within the network.
2) *Bandwidth-constrained.* Each peer in $\mathcal{I}$ expends only $O(\text{polylog}(N))$ bits in any round.

We assume that the total number of peers within the network is N in any round. As a best case scenario, all the peers are considered to be honest. We also assume that the amount of information required to uniquely represent a peer's network address is $\Omega(\log N)$ bits, i.e., there exists no encoding scheme to compress network addresses.

Public bulletin board. All the peers are given write access to a *bulletin board* or a linear log that provides read access to everyone (including the public) [18]. Each peer can write $O(\log N)$ bits to the board per write operation. However, there is a constraint that an arbitrary peer is selected to write to the board at every β rounds. More importantly, this bulletin board is the only interface through which the peers can disseminate information to the public.

Bootstrapping service. Since the system is dynamic, the peers must utilize the bulletin board to regularly update the peers that are responsible for new peers to join the network. We assume that the peers follow some algorithm to construct a bootstrapping service $\mathcal{I}$ using the bulletin board. For example, the peers may write their network addresses onto the bulletin board.

Joining the network. We also assume that there exists a join algorithm for new peers to contact the service and join the network. The minimum requirement for a new peer to join the network is to obtain a response from the bootstrapping service.

Theorem 5. *Any dynamic system that implements a bootstrapping service such as $\mathcal{I}$ using a public bulletin board, can support a half-life of only $\widetilde{\Omega}(\sqrt{\beta N})$.*

Proof sketch. For a new peer to obtain a response from the public bulletin board, it must send a message to a network address that is on that board. The network addresses in the most recent W bits of the bulletin board are considered to be part of the bootstrapping service. We show that information representing a network address is "useful" only if it is "recent".

No honest peer would be alive after $\Omega(\alpha \log N)$ rounds with a high probability. Recall that an honest peer fails with probability $1/2$ in any epoch. For some suitable constant c_1, after $c_1 \log N$ number of epochs, an honest peer would fail with a high probability. Therefore, if there is a network address that is $c_1 \alpha \log N$ rounds old, then there is a high probability that the peer controlling that network address has left. Since the system allows $O(\log N)$ bits to be written every β rounds, we get the following upper bound on W,

$$W = O(\alpha \log^2 N/\beta).$$

There can be other information in the most recent W bits; if there are more (distinct) network addresses, then the bootstrapping service can help in joining more new peers in a fixed period of time. Therefore, as a best case scenario, we assume that those W bits of information consists of only network addresses.

Let $\mathcal{U}$ be the highest number of join requests that a peer can handle. As a best case scenario, we assume that all the peers in the service are distinct, and each response message is equivalent to a successful join. Recall that in any α consecutive rounds, at least $N/2$ new peers must be able to join the network. Let K be the number of distinct network addresses in the most recent W bits on the board,

$$\alpha K \mathcal{U} \geq N/2.$$

If $\alpha = o\left(\sqrt{\frac{\beta N}{\mathcal{U} \log N}}\right)$, then in total, the system can handle much less than $N/2$ join requests in α consecutive rounds. ■

VIII. Conclusion and discussion

In this paper, we design protocols to maintain a dynamic hypercubic network despite high churn and the presence of

malicious peers. We show that the network formed by honest peers remains partition-resilient for a polynomial number of rounds with a high probability, even if the network size is allowed to vary by a polynomial factor. To the best of our knowledge, we construct the first secure bootstrapping service for a dynamic system consisting of bandwidth-constrained entities by exploiting the shared data on the blockchain. Moreover, we provide matching lower bounds for α if such a bootstrapping service is implemented using any public bulletin board.

Our results carry over to a setting where the churn adversary is slightly stronger. A $(0.5, \alpha)$-churn adversary can kill at most 0.5 fraction of honest peers in any α consecutive rounds. An α-late churn adversary can choose to kill an honest peer at some round r, but that honest peer actually leaves the network at $r + \alpha$ round. By making the lifetime of a directory node a constant times smaller than α/β blocks, the properties of the network (and lower bound for α) hold even in the presence of $(0.5, \alpha), \alpha$-late churn adversary. In the context of blockchains, committees do not require an honest majority, but this can also be done if $\rho < 0.5 - \epsilon$ for some small constant ϵ.

REFERENCES

[1] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in *24th {USENIX} Security Symposium ({USENIX} Security 15)*, 2015, pp. 129–144.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system (white paper)," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[3] Y. Marcus, E. Heilman, and S. Goldberg, "Low-resource eclipse attacks on ethereum's peer-to-peer network," Cryptology ePrint Archive, Report 2018/236, 2018, https://eprint.iacr.org/2018/236.

[4] K. Nayak, S. Kumar, A. Miller, and E. Shi, "Stubborn mining: Generalizing selfish mining and combining with an eclipse attack," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2016, pp. 305–320.

[5] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in *International conference on financial cryptography and data security*. Springer, 2014, pp. 436–454.

[6] A. Singh, M. Castro, P. Druschel, and A. Rowstron, "Defending against eclipse attacks on overlay networks," in *Proceedings of the 11th workshop on ACM SIGOPS European workshop*, 2004, pp. 21–es.

[7] A. Singh *et al.*, "Eclipse attacks on overlay networks: Threats and defenses," in *In IEEE INFOCOM*. Citeseer, 2006.

[8] J. Saia, A. Fiat, S. Gribble, A. R. Karlin, and S. Saroiu, "Dynamically fault-tolerant content addressable networks," in *International Workshop on Peer-to-Peer Systems*. Springer, 2002, pp. 270–279.

[9] K. Hildrum and J. Kubiatowicz, "Asymptotically efficient approaches to fault-tolerance in peer-to-peer networks," in *International Symposium on Distributed Computing*. Springer, 2003, pp. 321–336.

[10] M. Naor and U. Wieder, "Novel architectures for p2p applications: the continuous-discrete approach," *ACM Transactions on Algorithms (TALG)*, vol. 3, no. 3, pp. 34–es, 2007.

[11] A. Fiat, J. Saia, and M. Young, "Making chord robust to byzantine attacks," in *European Symposium on Algorithms*. Springer, 2005, pp. 803–814.

[12] B. Awerbuch and C. Scheideler, "Group spreading: A protocol for provably secure distributed name service," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2004, pp. 183–195.

[13] ——, "Towards a scalable and robust dht," *Theory of Computing Systems*, vol. 45, no. 2, pp. 234–260, 2009.

[14] R. Guerraoui, F. Huc, and A.-M. Kermarrec, "Highly dynamic distributed computing with byzantine failures," in *Proceedings of the 2013 ACM symposium on Principles of distributed computing*, 2013, pp. 176–183.

[15] M. Knoll, A. Wacker, G. Schiele, and T. Weis, "Bootstrapping in peer-to-peer systems," in *2008 14th IEEE International Conference on Parallel and Distributed Systems*. Ieee, 2008, pp. 271–278.

[16] T. Neudecker and H. Hartenstein, "Network layer aspects of permissionless blockchains," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 838–857, 2018.

[17] A. F. Loe and E. A. Quaglia, "You shall not join: A measurement study of cryptocurrency peer-to-peer bootstrapping techniques," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2231–2247.

[18] A. R. Choudhuri, M. Green, A. Jain, G. Kaptchuk, and I. Miers, "Fairness in an unfair world: Fair multiparty computation from public bulletin boards," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 719–728.

[19] *Blockchain.com Explorer*, 2020 (accessed August 28, 2020). [Online]. Available: https://www.blockchain.com/explorer

[20] *Etherchain - The Ethereum Blockchain Explorer*, 2020 (accessed August 28, 2020). [Online]. Available: https://etherchain.org/

[21] *BitInfoCharts - Bitcoin Explorer*, 2020 (accessed August 28, 2020). [Online]. Available: https://bitinfocharts.com/bitcoin/explorer/

[22] E. Bortnikov, M. Gurevich, I. Keidar, G. Kliot, and A. Shraer, "Brahms: Byzantine resilient random membership sampling," *Computer Networks*, vol. 53, no. 13, pp. 2340–2359, 2009.

[23] A. Fiat and J. Saia, "Censorship resistant peer-to-peer content addressable networks." in *SODA*, vol. 2, 2002, pp. 94–103.

[24] M. Naor and U. Wieder, "A simple fault tolerant distributed hash table," in *International Workshop on Peer-to-Peer Systems*. Springer, 2003, pp. 88–97.

[25] H. Johansen, A. Allavena, and R. Van Renesse, "Fireflies: scalable support for intrusion-tolerant network overlays," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 4, pp. 3–13, 2006.

[26] D. Dolev, E. N. Hoch, and R. Van Renesse, "Self-stabilizing and byzantine-tolerant overlay network," in *International Conference On Principles Of Distributed Systems*. Springer, 2007, pp. 343–357.

[27] H. D. Johansen, R. V. Renesse, Y. Vigfusson, and D. Johansen, "Fireflies: A secure and scalable membership and gossip service," *ACM Transactions on Computer Systems (TOCS)*, vol. 33, no. 2, pp. 1–32, 2015.

[28] 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*, vol. 31, no. 4, pp. 149–160, 2001.

[29] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking bitcoin: Routing attacks on cryptocurrencies," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 375–392.

[30] M. Tran, I. Choi, G. J. Moon, A. V. Vu, and M. S. Kang, "A stealthier partitioning attack against bitcoin peer-to-peer network," in *IEEE Symposium on Security and Privacy (S&P)*, 2020.

[31] E. Rohrer and F. Tschorsch, "Kadcast: A structured approach to broadcast in blockchain networks," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, 2019, pp. 199–213.

[32] Y. Mao, S. Deb, S. B. Venkatakrishnan, S. Kannan, and K. Srinivasan, "Perigee: Efficient peer-to-peer network design for blockchains," in *Proceedings of the 39th Symposium on Principles of Distributed Computing*, 2020, pp. 428–437.

[33] 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.

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

[35] R. Pass and E. Shi, "Fruitchains: A fair blockchain," in *Proceedings of the ACM Symposium on Principles of Distributed Computing*, 2017, pp. 315–324.

[36] D. Liben-Nowell, H. Balakrishnan, and D. Karger, "Analysis of the evolution of peer-to-peer systems," in *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, 2002, pp. 233–242.

[37] V. T. Aradhya, S. Gilbert, and A. Hobor, "A partition resilient overlay network via blockchain," 2020 (Accessed: January 13, 2020). [Online]. Available: https://www.comp.nus.edu.sg/∼varadhya/papers/AGH21-TechReport.pdf