Robust Overlays Meet Blockchains: On Handling High Churn and Catastrophic Failures[⋆]

Vijeth Aradhya[1(✉)][0009−0004−7982−1908], Seth Gilbert[1], and Aquinas Hobor[2,1]

[1] National University of Singapore, Singapore
{varadhya,seth.gilbert}@comp.nus.edu.sg
[2] University College London, United Kingdom
a.hobor@cs.ucl.ac.uk

Abstract. Blockchains have become ubiquitous in the world of robust decentralized applications. A crucial requirement for implementing a blockchain is a reliable "overlay network" providing robust communication among the participants. In this work, we provide communication-efficient and churn-optimal (barring log factors) Byzantine-resilient algorithms for maintaining blockchain networks. Our approach utilizes an interesting "cross-layer optimization" wherein the overlay network relies on the blockchain that is built on top of it. An important contribution is a tight "half-life" analysis on the *amount of churn* that can be tolerated, where peers have bandwidth restrictions. Moreover, by leveraging synergies between the blockchain and the overlay network, we can provide non-trivial recovery guarantees from unexpected *catastrophic failures*, which include a large class of connectivity issues such as denial-of-service, or exponentially unlikely lucky streaks for Byzantine peers, etc.

1 Introduction

A network formed by *logical* links among entities, wherein a logical link may consist of many *physical* links, is called an overlay network. Blockchains, distributed ledgers, and most other distributed services rely on overlays to facilitate communication. For example, cryptocurrencies such as Bitcoin [25] rely on a peer-to-peer network for fast and efficient communication among the peers.

However, existing overlay networks used by blockchains are insecure. In recent years, there have been attacks on the network connectivity provided by overlays, exploiting several aspects such as unsafe peer storage and connection policies [15, 22], weak network synchronization [33], and churn [34]. Moreover, such network partitioning attacks form the basis of other powerful attacks such as double spending, reducing effective honest resources, and selfish mining [10, 28].

In this work, we make connections between blockchains and robust overlays for distributed hash table, resulting in improvements for both systems, including efficient joining, guaranteed reliability, and the ability to recover from

[⋆] V. Aradhya and S. Gilbert acknowledge the support by Singapore MOE Tier 2 Grant MOE-T2EP20122-0014.

Table 1. Comparison under different performance metrics.

	Join latency (in rounds)	Join comm. complexity	Network size variation	Recovery (catastr-ophic failures)
Group spreading [4]	$O(\log N)$	$O(\log^3 N)$	✗	✗
S-Chord [12]	$O(\log N)$	$O(\log^3 N)$	✗	✗
Cuckoo rule [5]	$O(\log N)$	$O(\log^3 N)$	✗	✗
NOW [14]	$O(\log^4 N)$	$O(\log^6 N)$	✓	✗
This work	$O(1)$	$O(\log^3 N)$	✓	✓

bad situations. We exploit a form of "cross-layer optimization", where the overlay maintenance protocols use the blockchain for global coordination, and the blockchain, in turn, uses the efficient overlay for its communication. We provide a new Byzantine-resilient algorithm for maintaining an efficient overlay that tolerates near-optimal rates of churn. There are four key aspects in which our work improves on existing solutions by augmenting the overlay with a blockchain.

1. **Joining.** Byzantine peers can strategically join and leave to affect the overlay connectivity. This leads to Byzantine join-leave attacks [4], where Byzantine peers repeatedly rejoin the system. Thus, the join protocol is crucial to maintain the properties of the overlay. Unfortunately, existing solutions to such join-leave attacks can be expensive, both from latency and message complexity perspective [4,5,12,14].
2. **Bootstrapping and Churn Analysis.** Often, the problem of securely introducing a new peer to the system is unfortunately swept under the hood (both in theory and practice). This is, in fact, crucial to the robustness of join algorithms. This is also a reason for the lack of a thorough analysis on the churn tolerance of a system. A fundamental question in dynamic systems, where peers can join and leave, is how long do peers necessarily have to remain in the system so that it works properly? We answer this question in the context of blockchain networks, where the peers have limited bandwidth. We provide a "half-life" analysis of churn, showing that our overlay design achieves near-optimal churn.
3. **Changes in Network Size.** Early solutions [4,5,12] for join-leave attacks assumed that the network size changes by at most a constant factor. Those solutions maintained highly structured (routable) topologies, making it hard to shrink and expand the overlay if the network size were to polynomially change over time. A crucial property of the design in [14] is being able to efficiently adapt to such changes. We show that the peers can consistently be in consensus on key parameters of the overlay (via the blockchain), which results in a simple and efficient way to adapt to changes in network size.
4. **Recovery.** Existing solutions are "brittle" in that if Byzantine peers overwhelm a part of the overlay, then it is difficult to recover the original properties of the overlay. This situation can occur over time as these protocols (with probabilistic guarantees) are run indefinitely. Moreover, open distributed sys-

tems are vulnerable to denial-of-service attacks, for e.g., the Gnutella network, while resilient to random failures, could be split into a large number of disconnected components after a targeted attack [36]. We combine several ideas, i.e., limited lifetime for a peer [4], fault-tolerant topologies [8], and blockchain consensus, to quickly and efficiently recover from a large class of connectivity issues, termed as "catastrophic failures" (cf. Section 6).

Our Approach

The standard approach for fault tolerance in overlays is *replication* (e.g., [5, 11, 27]). Thus, our starting point is a virtual network, specifically a hypercube, in which each vertex of the hypercube is implemented in a replicated fashion by a small set of peers which are collectively termed as a *committee*. Such replication ensures that there are sufficient number of honest peers in each committee.

Typically, replication in dynamic overlays subjected to churn and Byzantine faults requires considerable coordination among peers. Our insight is that this coordination can be solved via the blockchain itself by storing small amounts of auxiliary data on the blocks to achieve the necessary synchronization. For e.g., our committees do not need to run a consensus algorithm, perform random walks, or do any other sort of coordination (unlike in [4, 5, 12, 14]). This is similar to a recent trend exploiting on-chain information to simplify and facilitate off-chain distributed algorithms, see, e.g., payment channel networks (e.g., [31]) or dynamic sharding [38].

A key observation is that blockchains are publicly available, i.e., their contents can be read by anyone. Specifically, a recent copy of the blockchain is available at all times, and this provides an entry point to the service (e.g., blockchain explorers [6, 7]). This allows new peers to easily join the network, avoiding more centralized solutions [20]. This paves the way for *explicitly* designing a secure and dynamic bootstrapping service that tolerates churn and Byzantine faults.

An existing model that captures a similar idea is a "public bulletin board" [24]. A blockchain differs from a typical bulletin board in three ways: (1) the amount of auxiliary information in a block must be small, (2) the rate at which information can be shared is limited by the block interval, and (3) the network may never be fully synchronized where each peer holds the same chain. Thus, one of our contributions is to carefully distill the properties provided by the blockchain in a way that the overlay algorithms can be concisely described while not losing track of real-world implementations.

Our goal is to *minimally* use the space on a block for maintaining the overlay. Specifically, each block stores the identity of only one peer in the overlay. To maintain a virtual hypercube with at most N peers, we rely on a set of about $\widetilde{\Theta}(\sqrt{N})$ of the most recent blocks which store a set of about $\widetilde{\Theta}(\sqrt{N})$ peers, which we refer to collectively as a *directory*. Each peer in the directory is responsible for a subset of the committees, and keeps track of the members of those committees.

If the adversary can generate unlimited (Sybil) identities, and if the system has churn, then over time, the adversary can take over honest peers' connections. As in many blockchains, we rely on proof-of-work to mitigate such attacks. The peers can *simultaneously* mine both identities and blocks *without* spending extra computational resources by using the 2-for-1 PoW technique [13, 30]. While we focus on proof-of-work for concreteness, our overlay design can similarly be made to work with blockchains based on a different resource constraint.

Churn. A primary goal of this paper is to understand the limits of the *rate* of churn. We adopt the *half-life* approach to churn rate for honest peers: if there are H peers in the system, then over a specific interval of time, the half-life, at most $H/2$ new peers can join or at most $H/2$ peers can depart. This allows for highly bursty behavior, with large numbers of concurrent joins and departures. We provide a lower bound for the feasible half-life that depends on the rate of churn and the bandwidth constraints.

As new blocks are added to the blockchain, and as existing blocks age, the members of the directory change, handing off information from old directory members to new directory members in a controlled process. Similarly, when the number of peers changes significantly, the size of the virtual hypercube has to change, migrating information to new directory members. Such information exchanges need to be carefully managed to avoid Byzantine interference.

Recovery. Finally, the blockchain is not just an alternative interface for new peers to join, it also aids the overlay to recover from catastrophic failures (e.g., a constant fraction of committees and their corresponding directory members are instantly corrupted). As long as most of the committees and directory are still functioning properly, our observation is that the overlay operates sufficiently well to continue installing new blocks, to continue replacing directory and committee members, and restoring the fully correct operation of the overlay, i.e., to ensure that there are again sufficient number of honest peers in every committee. To show recovery properties, we not only rely on fault-tolerant properties of the overlay topology (e.g., [3,8,11]), but also prove that the overlay can reliably and efficiently adapt to large network size variations *during* recovery.

Summary. We exploit the blockchain for securely bootstrapping peers and (global) coordination among peers (e.g., agreement on new topology, etc.). We summarize our contributions (that hold with high probability), in the context of a network with at most N peers, where the average block interval is β.

1. We design protocols to maintain a hypercubic overlay of committees for a polynomial number of rounds, where the half-life is $\alpha = \Theta(\beta\sqrt{N}\log N)$, the network size can vary polynomially over time, and each peer sends/receives only $O(\log^3 N)$ messages per round.
2. We show that when catastrophic failures occur, the overlay recovers (i.e., retains its original properties) within a constant number of half-lives.
3. We give a lower bound, barring log-factors, for half-life, $\alpha = \widetilde{\Omega}(\sqrt{\beta N})$, showing that it is impossible to tolerate higher rates of churn, even if peers share a public bulletin board that can be used for joining.

2 Related Works

An in-depth related work exposition can be found in the full version [1].

Early Designs. Fiat and Saia [11] design an overlay based on bipartite expanders where at least $(1-\epsilon)N$ peers can efficiently communicate with at least $(1-\epsilon)N$ peers for a small constant ϵ. Their topology is fixed; [35] modified it to handle a restricted form of churn. Datar [8] built a content addressable network over multi-hypercube, having fault-tolerance against adversarial deletion similar to [11], improving on the communication and storage costs. However, the design is not resilient against Byzantine failures. Many overlays were designed to be robust against random failures, where each peer (independently) has a bounded probability of being Byzantine [9, 16, 18, 26].
Join-Leave Attacks. DHTs are typically made robust by replication of each data item over a small group, e.g., a logarithmic number of peers, of which a majority are honest [11, 16, 26, 35]. Alas, Byzantine peers can often repeatedly join and leave until they overwhelm a particular group.

Awerbuch and Scheideler [4] showed that a hypercubic topology can be maintained (with logarithmic redundancy and honest majority) over polynomial number of join/leave events, where each peer simulates $O(\log N)$ nodes and every node has a limited lifetime, if at most $O(1/\log N)$ fraction of the nodes can be Byzantine. Subsequent works tolerate a linear fraction of Byzantine faults. Fiat et al. [12] used the k-rotation rule [37]; Awerbuch and Scheideler [5] introduced the cuckoo rule; Guerraoui et al. [14] exploit random walks to maintain clusters. Jaiyeola et al. [17] use the limited lifetime method and $O(\log\log N)$ redundancy to show that at least $(1-o(1))N$ peers can reach at least $(1-o(1))N$ peers. This line of work either assumes static "gateway peers" with unlimited bandwidth or access to random peers for bootstrapping new peers. Moreover, their join algorithms are rather complicated and expensive (see Table 1).

Recently, Augustine et al. [2] designed a Byzantine-resilient overlay network (for a fixed network size) using the idea of a "dynamic whiteboard" to incorporate new peers into the system. But their algorithms cannot be adapted in our case, which is optimized for integration with blockchains. They use a constant-degree expander topology; whereas we rely on a (fault-tolerant) routable topology, and prove that the overlay can recover from catastrophic failures.
Blockchain Overlays. Kadcast [32] builds on Kademlia [23] and proposes a structured broadcast protocol for disseminating blocks. It is unclear how this protocol performs with respect to high churn and Byantine faults. In Perigree [21], a peer retains the "best" subset of neighbours after regular intervals, and connects to a small set of random peers to explore potentially better-connected peers. But Perigree may actually be prone to eclipse attacks because the adversary can monopolize a peer's connections by providing well-connected neighbors.

3 Model

Entities. A peer is a real-world entity, and can be: (1) honest, following the protocol, or (2) Byzantine, arbitrarily deviating from the protocol. A peer can control multiple *identities*. The *network size* is the total number of peers. The *maximum* network size is denoted by N. Byzantine peers always constitute at most a ρ fraction of the network size. They know the network topology at any time (but they cannot modify or delete messages sent by honest peers).

Communication. Each peer maintains a set of *neighbouring* peers that it is *connected* with. The system proceeds in synchronous *rounds*; in each round, a message that is sent at the beginning of a round by a peer is assumed to reach its neighbours by the end of that round.

Computational Restriction. The peers have a *hash power* constraint, i.e., each peer owns one unit of hash power that allows the peer to query a hash function (modelled as a random oracle) q times in a round [13, 29]. (If an entity has more hash power, then it is viewed as a coalition of peers.) For any given input, the hash function provides a (fixed) random output of length $\kappa = \Theta(\log N)$.

Blockchain. If the overlay has at least $\mu_n(1-\rho)n$ honest peers[3] with a diameter of at most $2\log N$, and there are at most ρn Byzantine peers, for appropriate constants $\rho < \mu_n < 1$, where n is the *current* number of peers, then the blockchain is guaranteed to provide the following properties [29, 30] for honest peers. (In this work, our goal is to provably maintain an overlay with such properties.)

Safety. There exists a *confirmed chain*[4] C_u^r for any honest peer u in round r having the following properties: (i) Δ-Synchronization: If $|C_u^r|$ is honest peer u's confirmed chain length at round r, then by round $r+\Delta$, every honest peer's confirmed chain length is at least $|C_u^r|$; (ii) Consistency: C_u^r is a prefix of $C_u^{r'}$ for any round $r' \geq r$. At any round r, if $|C_u^r| \leq |C_v^r|$, then C_u^r is a prefix of C_v^r, and for a large enough constant μ_s, $|C_u^r| - |C_v^r| \leq \mu_s$.

Liveness. For large enough T and constant μ_b, any consecutive $T \geq \Omega(1)$ blocks are included in any confirmed chain in $[T\beta/\mu_b, T\mu_b\beta]$ rounds; β is the average block interval.

δ-Approximate Fairness. Any set of honest peers controlling a ϕ fraction of hash power is guaranteed to get at least a $(1-\delta)\phi$ fraction of the blocks in any $\Omega(\kappa/\delta)$ length segment of the chain.

Public Availability. There exists an introduction service $\mathcal{I}$ that provides a local copy of an existing honest peer's chain.

Churn. We consider the standard "partially oblivious" adversary [4, 5, 12, 14] that specifies the join/leave sequence σ for honest peers in advance. However, it can choose to adaptively join/leave Byzantine peers. In particular, after the first i events in σ are executed, the adversary can either choose to join/leave a Byzantine peer or initiate the $(i+1)^{\text{th}}$ event in σ.

Churn Rate. We consider the *half-life* measure [19] to model the churn rate for honest peers. 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; the doubling time is the number of rounds required for the number of honest peers to double. An *epoch*, denoted by α, is defined as the smallest halving time or doubling time over all rounds in the execution. Furthermore,

[3] We consider a (large) subset of honest peers because the network can get split into multiple components during a catastrophic failure (cf. Section 6).

[4] The interpretation of confirmed chain is typically blockchain-specific; for example, Bitcoin deems a block to be confirmed if it is at least 6 blocks deep.

we assume that the epoch is much greater than the average block generation interval; in other words, $\alpha \gg \beta \log N$.
Change of Network Size. The network size can change by a factor of at most 2 in any epoch. It can thus polynomially vary over time; the number of peers at any round r, $N_r \in [N^{1/y}, N]$ for any constant $y > 1$.

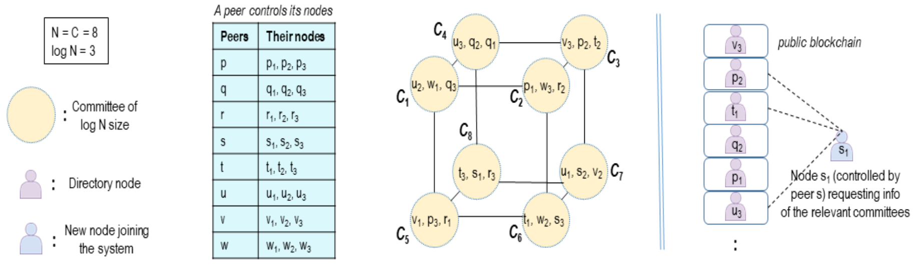
4 Overlay Design and Algorithms (Stable Network Size)

We first describe our overlay algorithm for a fixed network size of $\Theta(N)$. Two key parameters are the epoch length α (half-life) and the "directory size" $\mathcal{B}$. We will observe that α should roughly be the time it takes $\tilde{\Theta}(\sqrt{N})$ blocks to be added, and $\mathcal{B}$ (number of blocks in a directory) should be about $\tilde{\Theta}(\sqrt{N})$.
Nodes. Each peer generates and controls $\Theta(\log N)$ (virtual) identities, known as *nodes* in the overlay. Each peer participates (sends and receives messages) via 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.
Directory. Every node is initially a "non-directory" node. Some nodes also become directory nodes: a peer that adds a block to the blockchain promotes one of its nodes to a directory node, adding the identity of the promoted directory node, along with its network address, to the new block.
Committees. We maintain a hypercubic network of *committees*; each vertex corresponds to a committee. There are $\mathcal{C} = N$ committees, each consisting of $\Theta(\log N)$ nodes. If two committees are adjacent in the hypercube, then every pair of nodes in those committees are neighbors. Logarithmic redundancy ensures that a sufficient fraction of peers are honest so that the hypercube structure is maintained. Each committee is identified by a (unique) committee ID. As a peer may control multiple nodes, it may be present in multiple committees.
Limited Lifetime. Byzantine peers can repeatedly rejoin until their nodes get placed (by chance) in desired committees, potentially resulting in honest nodes having mostly Byzantine neighbors. A standard solution is to limit the lifetime of nodes, forcing them to rejoin (e.g., [4]). To avoid too much induced churn, the lifetime should be large, but not so large as to allow Byzantine join-leave attacks. Thus, we set the node lifetime to be $\Theta(\alpha/\beta)$ blocks, i.e., a constant number of half-lives. This ensures that at most a constant fraction of any honest peer's neighbours are Byzantine. Moreover, limited node lifetime is important for providing time bounds on recovery from catastrophic failures (see Section 6).

We now provide further details on the overlay maintenance. The technical difficulty lies in enabling directory nodes to help honest nodes join, while respecting the bandwidth constraints—even as the adversary may try to flood some directory nodes with requests. Figure 1 gives an overview of our design.

Directories

A directory on the blockchain comprises of $\mathcal{B}$ consecutive "buckets", and each bucket consists of consecutive $\lambda_d \log^2 N$ blocks, where λ_d is a suitable constant.



Peers	Their nodes
p	p_1, p_2, p_3
q	q_1, q_2, q_3
r	r_1, r_2, r_3
s	s_1, s_2, s_3
t	t_1, t_2, t_3
u	u_1, u_2, u_3
v	v_1, v_2, v_3
w	w_1, w_2, w_3

Fig. 1. High-level overview of our design for 8 peers.

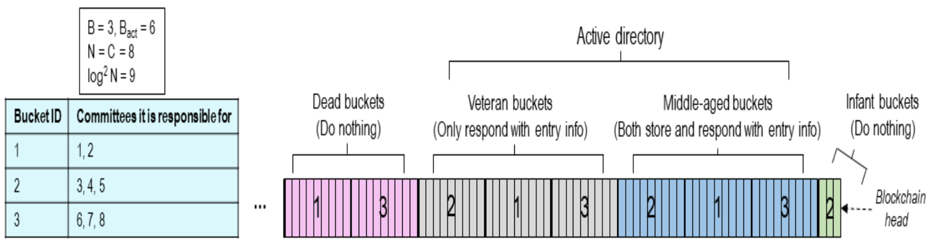
Each block refers to one directory node, so a directory refers to $\mathcal{K} = \mathcal{B}\lambda_d \log^2 N$ directory nodes. (The chain is divided into buckets from the beginning.)

There must always be enough honest directory nodes in each bucket, despite churn. By the blockchain fairness assumption, $\Theta(\log^2 N)$ bucket size is sufficient to ensure $\Omega(\log N)$ honest peers per bucket at any time.

Directory Responsibilities. Each bucket is *responsible* for a set of committees, i.e., directory nodes in that bucket help new nodes join those committees. There are two main *functions* of a directory node. (1) A directory node *stores* information about the nodes in its committees. (2) A directory node *sends* that information to new joining nodes and to neighboring committees. There exists a predetermined mapping $[\mathcal{C}] \to [\mathcal{B}]$ from committees to buckets, specifying which bucket manages a given committee, such that each bucket is responsible for (almost) the same number of committees.

Phases of a Bucket. We describe the phases of buckets and its directory nodes (and the transitions between phases), which determine the nodes' functions over time: (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. The directory nodes neither store entry information, nor respond to any requests. If all $\lambda_d \log^2 N$ blocks of the bucket are confirmed in the blockchain, then the bucket transitions to middle-aged. (2) *Middle-aged.* These directory nodes store new node entry information as they receive it, and reply to requests. If the bucket is not among the most recent (confirmed) $\mathcal{B}$ buckets, then the bucket transitions to veteran phase. (3) *Veteran.* These directory nodes do not store new nodes' entry information, but they do reply to requests with committee information already known to them. If the bucket is not among the most recent (confirmed) $\mathcal{B}_{act}$ buckets (to be defined), then it shifts to dead phase. (4) *Dead.* These directory nodes (as in the infant phase) neither store any new node information, nor respond to requests. During the transitions, there is a delay of Δ rounds to ensure that the confirmed chains of honest peers reach the same required height.

Active Directory. The *active directory*, also known as *bootstrapping service*, is the most recent $\mathcal{B}_{act}$ consecutive (confirmed) buckets. The most recent $\mathcal{B}$ consecutive buckets, forming an entire directory, are middle-aged. The rest of the buckets, forming one or more directories, are veteran. The number of blocks



Bucket ID	Committees it is responsible for
1	1, 2
2	3, 4, 5
3	6, 7, 8

Fig. 2. An illustration of buckets (and their blocks with directory nodes), represented by their IDs, in different phases.

and number of buckets in an active directory are denoted as $\mathcal{K}_{act}$ and $\mathcal{B}_{act}$ respectively. An example of an active directory is illustrated in Figure 2.

New nodes figure out the sequence of buckets in the active directory (using the blockchain and committee-directory mapping), and contact the relevant buckets to get the required committees' entry information for joining the network.

Node Joins and Lifetimes

A new node must provide a *proof-of-work* for joining the network, and directory nodes only interact with a new node if its proof is valid.

Proof for Joining. Let N_c be a nonce, $\hat{\mathrm{B}}_l$ be the hash of the latest confirmed block B_l, and *net_addr* be the network address of the peer. Then, the peer evaluates, $P_{join} = \mathbf{H}(\hat{\mathrm{B}}_l \parallel net\_addr \parallel N_c)$, where $\mathbf{H}$ is the (pseudorandom) hash function, to join the network through a new node. If $P_{join} < T_{join}$, where T_{join} is the *mining target* for joining, then the node is said to be a "valid" node, which means that the peer would be able to communicate with the bootstrapping service to register that node, and join the network. A directory node rejects the proof if B_l is not among the most recent μ_s blocks in its confirmed chain.

Joining the Network. A node's *entry information* constitutes its network address, the nonce N_c and the block number of the block that was used while mining for that node. We now describe the steps taken by a peer p to generate and join a new node q into the network. (1) It first produces a proof for joining. The leftmost $\log N$ bits of P_{join} represent the ID of the (random) committee, denoted by c, to which this new node would belong to. Let C_{rel} be the set of committees neighboring c in the hypercube, including c. (2) Peer p sends its entry information to all directory nodes in the middle-aged bucket responsible for committee c. (3) Peer p requests information, sampling $\Theta(\log N)$ nodes uniformly in each bucket responsible for committee c and requesting information on each of its neighbors in the hypercube. The directory nodes respond with the relevant entry information. (4) Peer p appropriately sends messages to handle Δ-synchrony. Let b_1 and b_2 be the first and $(\mathcal{B}+1)^{\text{th}}$ confirmed buckets (i.e., most recent middle-aged and veteran buckets). If the number of blocks confirmed after bucket b_1 is at most μ_s, and if b_1 is responsible for committee c, then p send its entry information to all nodes in b_2. (5) To complete the join, node q takes the union of valid entry information received in responses (as the adversary can only under-represent the nodes in a committee). It then sends its entry information to the nodes in each neighboring committee in the hypercube.

The key observation is that a constant fraction of directory nodes in a bucket are honest and available at any time, due to logarithmic redundancy in buckets and blockchain fairness. Thus, it is sufficient for the new node needs to hear back information from $O(\log N)$ directory nodes in each bucket (Step 3), reducing the communication complexity of join down to $O(\log^3 N)$, i.e., the new node contacts at most $\log N$ buckets, wherein $O(\log N)$ directory nodes in each bucket reply with committee information of $O(\log N)$ nodes.

Lifetime of Non-directory Node. The lifetime of a non-directory node is $T_l = \Theta(\alpha/\beta)$ blocks. The node u that had joined at block number b_l (which can be checked in its proof for joining) is considered invalid after block $b_l + T_l$ is confirmed, at which point the peer stops controlling that node u, and all other nodes that had node u as a neighbour remove u from their neighbour list.

Lifetime of Directory Node. If a node is promoted to a directory node, then it obtains another life (separate from the non-directory life). The directory node is considered to be alive for T_{dl} blocks from the block in which it is embedded in. We set T_{dl} to be $\Theta(\alpha/\beta)$ blocks, where $T_l < T_{dl}$. This time is chosen to be sufficiently long for the directory node to reach the veteran phase, and then for the lifetimes of all the non-directory nodes in its committee to fail (i.e., an additional T_l blocks), i.e., $T_{dl} > (1+\mathcal{B})\lambda_d \log^2 N + T_l$.

Node Generation Rate. The mining target T_{join} is set such that, in each epoch, the expected number of valid nodes that can be generated during an epoch is equal to $\Theta(N \log N)$. Each committee has $\Theta(\log N)$ nodes at any time because in every epoch, about $\Theta(N \log N)$ nodes join, while a similar number of them leave due to limited lifetime (set to be a constant number of epochs).

Directory Size and Half-Life (Relation between $\mathcal{B}$ and α)

First, the lifetime of a non-directory node is $\Theta(\alpha/\beta)$ blocks, so $\mathcal{B} \leq \Theta\left(\frac{\alpha}{\beta \log^2 N}\right)$, so that the node can fully participate in a directory's life cycle.

Due to node generation rate, the system must have bandwidth for $\Theta(N \log N)$ nodes to join in any α (consecutive) rounds. As a new node sends a join request to all directories within the active directory (both middle-aged and veteran), it suffices to focus on the number of join requests handled by one directory.

Due to the proof-of-work mechanism, we can ensure that the join requests are load-balanced across the rounds in an epoch. Let λ_{jr} be the highest number of join requests that can be handled by a bucket per round. For any directory, we calculate the total number of join requests that need to be handled in any round as $\mathcal{B}\lambda_{jr} \geq \Theta\left(\frac{\beta N \log^2 N}{\alpha}\right)$, where LHS is the total number of join requests that can be handled in a round, and RHS represents the minimum number of join requests that need to be handled in a round. The extra β factor is due to a possible precomputation attack, where an adversary sends join requests computed over the last μ_s confirmed blocks, and the extra $\log N$ factor is due to new nodes contacting $O(\log N)$ buckets while joining. These extra multiplicative factors ensure that the communication complexity per peer per round is $O(\log^3 N)$.

Ideally, we want to minimize α to handle maximum churn. Solving the above constraints, with bandwidth cost of $O(\log^3 N)$ messages per round for a peer, we find that $\alpha = \Theta(\beta\sqrt{N}\log N)$ and $\mathcal{B} = \Theta(\sqrt{N}/\log N)$. In Section 7, we show that this value of α is close to optimal.

Analysis Overview

We present the key lemmas and theorems for stable network size. The complete analysis can be found in the full version [1]. Consider a connectivity property called *partition resilience*, where (1) each committee has $O(\log N)$ nodes and $\Omega(\log N)$ honest peers, (2) every pair of honest nodes in each committee are connected, and (3) each honest node is connected to $\Omega(\log N)$ honest peers in each neighbouring committee. An epoch e is said to be *bandwidth-adequate* if each peer needs to send/receive $O(\log^3 N)$ messages for overlay maintenance in any round. The active directory is said to be *robust* if each bucket has $\Omega(\log^2 N)$ honest nodes that store entry information of all nodes in the relevant committees.

First, we see that no peer gets too many join requests due to load balancing, random committee assignment (via hash function), and random sampling:

Lemma 1. *Each peer receives $O(\log^2 N)$ join requests and information requests in any round in an epoch with high probability.*

Next, we argue that if the bandwidth is not exceeded, then the active directory is properly constructed and has the correct information to process join requests:

Lemma 2. *The active directory is* robust *in the epoch with high probability, if the last $\Theta(\beta T_{dl}/\alpha)$ epochs are* bandwidth-adequate.

Together, these lemmas imply that all joins are successful, ensuring the overlay is well-connected with each committee having sufficient number of honest peers:

Theorem 1. *The network is* partition-resilient *for polynomial number of rounds with high probability.*

5 Extension to Dynamic Network Size

We briefly discuss how to augment the protocols described in Section 4 to handle polynomial variation in network size over time. These techniques are an extension of the previous idea—see the full version [1] for the details.

The main problem caused by changing numbers of peers is that the system needs to adapt to maintain logarithmic redundancy. If the number of committees remains fixed, while the number of peers decreases, then each peer would need to simulate too many nodes at any time in order for the system to maintain $\Theta(\log N)$ nodes in every committee, exhausting the peer's bandwidth. And if the network size keeps increasing, then some peers may not be able to participate in the network all the time because they may take too long to generate nodes. Therefore, we adapt the size of the hypercube and the number of committees.

The key insight is that the blockchain can facilitate the necessary coordination to switch to a new topology. The first step is to efficiently estimate the network size, every constant number of epochs. Each node keeps track of new nodes that joined its committee in a span of fixed number of blocks. Then, all the nodes of a (random) committee (determined by a hash function) broadcast that count along with the entry information of new nodes that joined that committee (as evidence of the count) to everyone. This sampled change in committee size allows peers to estimate the size of the network. This information is then included on the blockchain to ensure that all peers agree on it.

Each peer then uses this estimate to determine whether the dimension of the hypercube is changing and number of committees in the new hypercube. Again, the information is placed on the blockchain to ensure agreement.

At that point, if necessary, the dimension of the hypercube is changed. During dimension change, the new hypercube is constructed while the old one is kept around to ensure connectivity. The directory goes into a "split state" for about one epoch, serving committees in the current hypercube and also constructing committees in the next hypercube. New nodes also get placed in the next hypercube, as well as the old one. Until each committee in the next hypercube has a sufficient number of new (honest) nodes, the overlay operates in the old hypercube. The directory then stops serving committees in the old hypercube, and the network adopts the new hypercube for broadcasting blocks.

6 Recovery

Catastrophic failures model a large class of connectivity issues, e.g., denial-of-service attacks or low probability events. We say that a *bucket fails* if $> 1/2$ of its honest peers are *corrupted*; a *committee fails* if it has $< 20 \log N$ honest peers, or if the bucket responsible for it has failed. The *corruption* could be Byzantine, or simply crashing, if honest peers leave faster than allowed by the churn model.

Consider an "(ϵ, δ)-catastrophic failure" where at most ϵ fraction of committees and/or buckets may have failed, and in total, at most δ fraction of honest peers get corrupted (for small constants ϵ and δ), while there still exists a connected network of honest peers of linear size and logarithmic diameter for which bootstrapping can be securely done. Such failures model exceptional scenarios that occur in practice wherein the network is split into multiple components, resulting in considerable wastage of honest peers' hash power over time.

Recovery. Our goal is to provably show that the network *recovers* from such catastrophic failures in a short period of time. Here, we naturally define recovery as the event at which the overlay retains its original properties.

There are two basic requirements for the overlay to recover from a catastrophic failure. First, a large fraction of honest peers can continue to run the blockchain protocol (ensuring the blockchain provides the same guarantees), albeit some honest peers may end up unable to participate fully (i.e., effective honest hash power is reduced) for a brief period of time. Secondly, the introduction service is not affected by the failure, i.e., it continues to return the chain of an honest peer in the largest connected component of honest peers.

The high-level intuition for recovery is to replace the entire active directory by a new one (that has no bucket failures), which will reliably facilitate new honest node joins. After a catastrophic failure, there is a large connected component of sufficient number of honest peers with a low diameter, that is responsible for the progress of the blockchain. First, this component should get replenished with new (honest) peers amidst churn, maintaining the required proportion. Then, the coordination protocols such as network size estimation, etc, must work for that component *during* recovery. Specifically, we rely on the fault-tolerance of the overlay topology [8] and properties of bipartite expanders [11] for showing that changing dimensions after a catastrophic failure does not inhibit recovery.

For a wide range of failures, the overlay, augmented with a blockchain, can exhibit a novel recovery property. The security arguments in existing designs for join-leave attacks [4,5,12,14,17] heavily rely on honest majority in committees. In the full version [1], we show the inherent difficulty of such localized algorithms to recover from committee failures, e.g., a small number of committees having malicious majority, can keep maintaining the majority over time.

Theorem 2. *If the network experiences (ϵ, δ)-catastrophic failure, then it becomes partition-resilient within $O(1)$ epochs with high probability.*

7 Lower Bound for Half-Life

In dynamic overlays, there is always a problem of bootstrapping a new peer, i.e., how does a new peer contact an existing peer within the network? A bootstrapping service $\mathcal{S}$ should have two properties: (i) *Secure:* the service responds with the identity of at least one honest peer in the network; and (ii) *Bandwidth-constrained:* Each peer communicates at most $\tilde{O}(1)$ bits in any round.

We show that there are unavoidable trade-offs in implementing such a service. Notably, the overlay algorithm described in this paper satisfies these basic service requirements—and so it is also subject to these inevitable trade-offs.

Let N be the number of peers in the network, and assume peers are honest. Assume network addresses require $\Omega(\log N)$ bits, i.e., there is no encoding scheme to compress network addresses. Peers can write $O(\log N)$ bit messages to a publicly visible *bulletin board* (e.g., [24]). An arbitrary peer is selected to write to the board at every β rounds. The bulletin board is the only interface through which the peers can disseminate information to the public.

We prove the following theorem by analyzing the number of bits of useful information on the bulletin board, compared to the number of identifiers needed to support the bandwidth required for all joins. Our theorem depends only on the minimum requirement that to complete a join operation, a newly joining node must receive at least one message from an existing member of the network.

Theorem 3. *Any dynamic system that implements a bootstrapping service $\mathcal{S}$ using a public bulletin board can support a half-life of only $\widetilde{\Omega}(\sqrt{\beta N})$.*

References

1. Aradhya, V., Gilbert, S., Hobor, A.: OverChain: Building a robust overlay with a blockchain. arXiv preprint arXiv:2201.12809 (2022)
2. Augustine, J., Bhat, W.G., Nair, S.: Plateau: A secure and scalable overlay network for large distributed trust applications. In: Stabilization, Safety, and Security of Distributed Systems: 24th International Symposium, SSS 2022, Clermont-Ferrand, France, November 15–17, 2022, Proceedings. pp. 69–83. Springer (2022)
3. Augustine, J., Chatterjee, S., Pandurangan, G.: A fully-distributed scalable peer-to-peer protocol for byzantine-resilient distributed hash tables. In: Proc. of the 34th ACM Symp. on Parallelism in Algorithms and Architectures. pp. 87–98 (2022)
4. Awerbuch, B., Scheideler, C.: Group spreading: A protocol for provably secure distributed name service. In: International Colloquium on Automata, Languages, and Programming. pp. 183–195. Springer (2004)
5. Awerbuch, B., Scheideler, C.: Towards a scalable and robust dht. Theory of Computing Systems **45**(2), 234–260 (2009)
6. BitInfoCharts: Bitcoin Explorer (2023), `https://bitinfocharts.com/bitcoin/explorer/`
7. Blockchain.com: Explorer (2023), `https://www.blockchain.com/explorer`
8. Datar, M.: Butterflies and peer-to-peer networks. In: European Symposium on Algorithms. pp. 310–322. Springer (2002)
9. Dolev, D., Hoch, E.N., Van Renesse, R.: Self-stabilizing and byzantine-tolerant overlay network. In: International Conference On Principles Of Distributed Systems. pp. 343–357. Springer (2007)
10. Eyal, I., Sirer, E.G.: Majority is not enough: Bitcoin mining is vulnerable. In: Intl. Conf. on Financial Cryptography and Data Security. pp. 436–454. Springer (2014)
11. Fiat, A., Saia, J.: Censorship resistant peer-to-peer networks. Theory of Computing **3**(1), 1–23 (2007), (previously appearing in SODA 2002)
12. Fiat, A., Saia, J., Young, M.: Making chord robust to byzantine attacks. In: European Symposium on Algorithms. pp. 803–814. Springer (2005)
13. Garay, J., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 281–310. Springer (2015)
14. Guerraoui, R., Huc, F., Kermarrec, A.M.: Highly dynamic distributed computing with byzantine failures. In: Proc. of the 2013 ACM Symposium on Principles of Distributed Computing. pp. 176–183 (2013)
15. Heilman, E., Kendler, A., Zohar, A., Goldberg, S.: Eclipse attacks on bitcoin's peer-to-peer network. In: 24th USENIX Security Symposium. pp. 129–144 (2015)
16. Hildrum, K., Kubiatowicz, J.: Asymptotically efficient approaches to fault-tolerance in peer-to-peer networks. In: International Symposium on Distributed Computing. pp. 321–336. Springer (2003)
17. Jaiyeola, M.O., Patron, K., Saia, J., Young, M., Zhou, Q.M.: Tiny groups tackle byzantine adversaries. In: 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS). pp. 1030–1039. IEEE (2018)
18. Johansen, H.D., Renesse, R.V., Vigfusson, Y., Johansen, D.: Fireflies: A secure and scalable membership and gossip service. ACM Transactions on Computer Systems (TOCS) **33**(2), 1–32 (2015)
19. Liben-Nowell, D., Balakrishnan, H., Karger, D.: Analysis of the evolution of peer-to-peer systems. In: Proceedings of the twenty-first annual symposium on Principles of distributed computing. pp. 233–242 (2002)

20. Loe, A.F., Quaglia, E.A.: You shall not join: A measurement study of cryptocurrency peer-to-peer bootstrapping techniques. In: Proc. of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 2231–2247 (2019)
21. Mao, Y., Deb, S., Venkatakrishnan, S.B., Kannan, S., Srinivasan, K.: Perigee: Efficient peer-to-peer network design for blockchains. In: Proceedings of the 39th Symposium on Principles of Distributed Computing. pp. 428–437 (2020)
22. Marcus, Y., Heilman, E., Goldberg, S.: Low-resource eclipse attacks on ethereum's peer-to-peer network. Cryptology ePrint Archive, Report 2018/236 (2018)
23. Maymounkov, P., Mazieres, D.: Kademlia: A peer-to-peer information system based on the xor metric. In: International Workshop on Peer-to-Peer Systems. pp. 53–65. Springer (2002)
24. Mitzenmacher, M.: How useful is old information? IEEE Transactions on Parallel and Distributed Systems **11**(1), 6–20 (2000)
25. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (White Paper) (2008), `https://bitcoin.org/bitcoin.pdf`
26. Naor, M., Wieder, U.: A simple fault tolerant distributed hash table. In: International Workshop on Peer-to-Peer Systems. pp. 88–97. Springer (2003)
27. Naor, M., Wieder, U.: Novel architectures for p2p applications: the continuous-discrete approach. ACM Transactions on Algorithms (TALG) **3**(3), 34–es (2007)
28. Nayak, K., Kumar, S., Miller, A., Shi, E.: Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In: 2016 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 305–320. IEEE (2016)
29. Pass, R., Seeman, L., Shelat, A.: Analysis of the blockchain protocol in asynchronous networks. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 643–673. Springer (2017)
30. Pass, R., Shi, E.: Fruitchains: A fair blockchain. In: Proceedings of the ACM Symposium on Principles of Distributed Computing. pp. 315–324 (2017)
31. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments (2016), `https://lightning.network/lightning-network-paper.pdf`
32. Rohrer, E., Tschorsch, F.: Kadcast: A structured approach to broadcast in blockchain networks. In: Proceedings of the 1st ACM Conference on Advances in Financial Technologies. pp. 199–213 (2019)
33. Saad, M., Anwar, A., Ravi, S., Mohaisen, D.: Revisiting nakamoto consensus in asynchronous networks: A comprehensive analysis of bitcoin safety and chainquality. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 988–1005 (2021)
34. Saad, M., Chen, S., Mohaisen, D.: Syncattack: Double-spending in bitcoin without mining power. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 1668–1685 (2021)
35. Saia, J., Fiat, A., Gribble, S., Karlin, A.R., Saroiu, S.: Dynamically fault-tolerant content addressable networks. In: International Workshop on Peer-to-Peer Systems. pp. 270–279. Springer (2002)
36. Saroiu, S., Gummadi, P.K., Gribble, S.D.: Measurement study of peer-to-peer file sharing systems. In: Multimedia Computing and Networking 2002. vol. 4673, pp. 156–170. International Society for Optics and Photonics (2001)
37. Scheideler, C.: How to spread adversarial nodes? Rotate! In: Proc. of the Thirty-Seventh Annual ACM Symposium on Theory of Computing. pp. 704–713 (2005)
38. Tennakoon, D., Gramoli, V.: Dynamic blockchain sharding. In: 5th International Symposium on Foundations and Applications of Blockchain 2022 (FAB 2022). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2022)