Good Things Come in LogLog(n)-Sized Packages

Robustness with Small Quorums

Mercy O. Jaiyeola[1], Kyle Patron[2], Jared Saia[3], Maxwell Young[1], and Qian M. Zhou[1]

[1]Mississippi State University, Dept. of Computer Science and Eng., Miss. State, MS, USA,
moj31@msstate.edu, myoung@cse.msstate.edu, qz70@msstate.edu
[2]Palantir Technologies, New York, USA, kyle.patron@gmail.com
[3]University of New Mexico, Department of Computer Science, Albuquerque, NM, USA,
saia@cs.unm.edu

May 31, 2017

Abstract

A popular technique for tolerating Byzantine faults in open distributed systems is to group machines into sets called ***quorums***, each of which has an honest majority. These quorums are then used as basic building blocks to design systems that are robust to adversarial faults.

Despite over a decade of active research, all current algorithms require quorum sizes of $\Omega(\log n)$, where n is the number of machines in the network. This size is important since communication cost scales polynomially in the size of the quorum. Given the stubbornness of this $\Omega(\log n)$ barrier, a natural question is whether better bounds are possible.

In this paper, we demonstrate that it is possible to reduce quorums sizes to $O(\log \log n)$, despite an adversary that controls a constant fraction of the computational resources in the network. In particular, we show that even with such small quorums, we can ensure that all but an $o(1)$-fraction of the machines can communicate with all but an $o(1)$-fraction of the machines in the network.

1 Introduction

Byzantine fault tolerance addresses the challenge of performing useful work when machines (***nodes***) in a system are malicious. Information routed through or stored at a faulty node can be discarded or corrupted, and tasks executed on such nodes may fail, or output an erroneous value. A popular technique for overcoming these challenges is to group nodes into sets called ***quorums***,[1] where each has a non-faulty majority. We can then ensure the following.

- Computation is performed by all members of a quorum via protocols for Byzantine agreement (BA) or more general secure multiparty computation to guarantee that tasks execute correctly. In this way, each quorum simulates a ***reliable processor*** upon which jobs can be run.

- Fault-tolerant routing is also ensured. For quorums Q_1 and Q_2 along a route, each member of Q_1 can transmit messages to all members of Q_2. This all-to-all exchange, followed by majority filtering by each node in Q_2, guarantees correctness of communication between quorums.[2]

[1]Such sets have appeared under different names in the literature, such as "swarms" [17] or "clusters" [20].

[2]Quorums also improve robustness in other ways. Members may agree to ignore another quorum if it misbehaves too often, hence reducing spamming. Data may also be redundantly stored at multiple quorum members.

The use of quorums provides a *scalable* approach to designing a robust distributed system, since they avoid the need to have all n nodes perform BA in concert, or communicate via a system-wide pairwise exchange of messages.

Quorums have been an active topic of research for more than a decade, with many intriguing theoretical results [7–9, 11, 17, 20, 21, 23, 27, 34, 36, 43, 49]. Yet, despite this progress, an enduring requirement is that each quorum contains $\Omega(\log n)$ nodes.

Why does this logarithmic term matter? At first glance, it is an unlikely bottleneck. However, since quorums are building blocks for the system, their size $|Q|$ greatly impacts important costs:

(i) *Cost of Quorum Computation.* Members of a quorum must often perform computation in concert. Consequently, resource costs are a function of $|Q|$. For example, employing BA [24–26] or secure MPC [12] protocols will require poly($|Q|$) messages.

(ii) *Cost of Robust Routing.* Routing via all-to-all exchange between two quorums incurs $\Omega(|Q|^2)$ message complexity. Given a route of length D, communication between any two quorums requires $O(D|Q|^2)$ messages.[3]

(iii) *Cost of State Maintenance.* Each node w must maintain state on all of its neighbors; this includes both the members of all quorums to which w belongs *and* the members of neighboring quorums. This requires storing link information, as well as periodically testing links for liveness. [4]

In each case above, reducing $|Q|$ would directly reduce cost. Unfortunately, for existing results, $|Q| = \Omega(\log n)$ is key to ensuring (via a concentration result and union bound) that all quorums have a non-faulty majority with high probability (w.h.p.).[5] Without this property, all previous quorum constructions succumb to adversarial attack. Therefore, new ideas are required to decrease $|Q|$.

1.1 ε-Robustness

Consider a system of n nodes where a β-fraction suffer Byzantine faults. The following defines our notion of ***ε-robustness***: *For a small $\varepsilon > 0$, at least $(1-\varepsilon)n$ quorums have an honest majority and can robustly route messages to each other.*

Note that prior results fall under this definition when $\varepsilon = 1/\text{poly}(n)$. For generality, the parameters β and ε are left unfixed, however, in order to employ BA protocols, typically $\beta \leq 1/4$, and a small ε is desirable. We consider the following questions:

Why is this a useful concept? Consider decentralized storage and retrieval of data. This definition guarantees all but an ε-fraction of data is reachable and maintained reliably. Example applications include distributed databases, name services, and content-sharing networks. Alternatively, consider n jobs in an open computing platform that are run on individual machines. This definition guarantees that all but an ε-fraction of those jobs can be correctly computed.[6]

Why isn't satisfying this definition trivial? Given $\Theta(n)$ non-faulty nodes, this definition captures the natural goal of simulating $(1-\varepsilon)n$ reliable processors *and* being able to route information between those processors. If we ignore the use of quorums or, equivalently, consider quorums each consisting of a single node, then

[3] We note that improvements are possible, but they come with certain caveats. Results in [17, 43] lower the cost to $O(D|Q|)$ in expectation but requires a non-trivial (expander-like) construction, and [49] further reduces this to $O(D)$ in expectation but with a poly($|Q|$) message cost each time routing tables are updated – this is expensive even with moderate churn.

[4] Constructions [17, 36] require that each node belongs to $\eta > 1$ quorums for a state overhead of $\Omega(|Q|\eta)$. Also, if each quorum shares links with Δ other quorums (they are neighbors), then $O(|Q|\Delta)$ such links must be maintained; typically, $\Delta = O(\log n)$.

[5] With probability at least $1 - 1/n^c$ for a tunable constant $c \geq 1$.

[6] While this may not be sufficient for general computation, it is valuable for tasks where an $o(1)$ error rate or bias can be tolerated; for example, if we wish to obtain statistics on a group of machines for network metrics, or generate a large set of random numbers.

we trivially have $(1-\beta)n$ reliable processors. However, routing between them is challenging. For example, establishing links between each pair of nodes will give robust routing, but this is hardly scalable.

Why don't previous solutions solve this problem? All prior results using quorums focus on the case where $\varepsilon = 1/\text{poly}(n)$. In this case, routing is possible – albeit, costly – because w.h.p. *all* quorums have a majority of honest nodes and are, therefore, reliable.

To reduce cost, we consider $\varepsilon = 1/\text{poly}(\log n)$ and refer to this as ***almost-everywhere routing***. This allows us to reduce the size of the quorum exponentially which yields cost savings. However, we lose the w.h.p. guarantee that all quorums have a majority of non-faulty nodes — indeed, w.h.p. there will be quorums that do not have such a majority— and this disqualifies prior solutions.

1.2 Related Work

Robustness via Quorums. The use of quorums for building robust distributed systems has received significant attention. Early results addressed robustness for $\varepsilon = 1/\text{poly}(n)$ with $\text{poly}(\log n)$ cost assuming constraints on the amount of dynamism in the system [5, 16, 17, 21, 35].

Full dynamism was achieved by Awerbuch and Scheideler in a series of breakthrough results [7–9]. In particular, the authors propose a *cuckoo rule* that w.h.p. preserves a good majority in each quorum over $n^{\Theta(1)}$ join/leave operations when the total system size remains $\Theta(n)$. More recently, Guerraoui et al. [20] showed similar guarantees for systems that can vary polynomially in size.

An experimental evaluation of the cuckoo rule, along with proposed improvements, is given in [45]. The trade-off between quorum size and the level of robustness is examined, and findings suggest the approach can be practical to a point. For example, when $n = 8,192$ and $\beta \approx 0.002$, under the original cuckoo rule, $|Q| = 64$ suffices to preserve a good majority in each quorum for $100,000$ join/departure events; $\beta \approx 0.07$ is possible with suggested improvements in [45].

Several results have focused on reducing communication complexity, when the goodness of quorums is guaranteed (via an algorithm like the cuckoo rule) [43, 49]. However, here too, quorum size impacts performance and $|Q| = 30$ incurs significant latency in practice [49]. Quorums have also been used in conjunction with quarantining Byzantine nodes [27, 41]; however, maintaining these quorums under churn remains an open problem.

Robustness without Quorums. Other decentralized robust constructions exist that do not explicitly use quorums [13, 16, 42]. However, the associated techniques retain some form of $\Omega(\log n)$ redundancy with regards to data placement or route selection and, therefore, incur $\text{poly}(\log n)$ cost.

Approaches described in [11, 23, 34] mitigate Byzantine faults by routing along multiple diverse routes. However, it is unclear that these systems can provide theoretical guarantees on robustness.

Central authorities (CAs) — sometimes referred to as a *Configuration Service* or *Neighborhood Authority* — have been used in prior results [11, 39, 40, 47] to achieve robustness. While our results can be used in conjunction with a CA, it is not always plausible to assume such an authority is available and immune to attack. For this reason, our work does not depend on a CA.

Computational Puzzles. Proof-of-work (PoW) via computational puzzles has been used to mitigate the Sybil attack [14] whereby an adversary overwhelms a system with a large number of identifiers (IDs). We note that such PoW schemes have been proposed in decentralized settings such as DHTs (for example, see [29]). However, such PoW schemes only *limit the number of Sybil IDs* — typically commensurate with the amount of computational power available to the adversary — and the problem of tolerating these adversarial IDs must still be addressed by other means (for example, see [44, 50]).

A prominent example of how PoW can provide security is Bitcoin. However, note that the analysis of Bitcoin and related systems commonly assumes the existence of a communication protocol that allows a node to disseminate a value to all other nodes within a known bounded constant amount of time despite an adversary [19, 30, 32]. In contrast, our results do not assume the existence of such a protocol. Our work does, however, use computational puzzles to obtain a significant reduction in communication complexity and state maintenance.

1.3 Our Model and Preliminaries

A node is ***good*** if it obeys protocol, otherwise, the node is Byzantine or ***bad***. For ease of exposition, we analyze a system where n nodes are always present even under churn; that is, when a node leaves, another is assumed to join. Our results hold when the system size is $\Theta(n)$ – that is, the size changes by a constant factor – but we omit these details in this extended abstract.

The Adversary. At most βn nodes are bad and used by an adversary to attack the system where $\beta < 1/4$ is a positive constant. Critically, this implies robustness to $\Theta(n)$ Byzantine nodes which is asymptotically optimal.[7] Note that this is a powerful attack model since a single adversary allows the bad nodes to perfectly collude and coordinate their malicious actions. The adversary also knows the full network topology and the contents of all messages sent between nodes; however, the adversary does not have access to any random bits generated by a good node.

Quorums. A ***quorum*** is a set of nodes; for our results, we assume each quorum has size $\Theta(\log \log n)$. Each node w has its own quorum Q_w and w is referred to as the ***leader***. A quorum Q is ***good*** if (i) $d_1 \ln \ln n \leq |Q| \leq d_2 \ln \ln n$ for sufficiently large positive constants $d_1 < d_2$, and (ii) the number of bad nodes in Q is at most $(1+\delta)\beta|Q|$ for some tunably small constant $\delta > 0$ depending only on n. Note that quorums are not necessarily disjoint; in addition to being the leader of Q_w, node w may belong to other quorums. Quorum construction is described in Section 3.1.

Input Graph. We assume an ***input graph***, G on N nodes.[8] Given that each node has an ID u.a.r. in $[0, 1)$ and there is no adversary, G satisfies the following properties with probability at least $1 - N^{-c}$ for a tunable constant $c \geq 1$:

- *P1 – Search Functionality.* There exists a ***search*** algorithm that, for any ***key value*** x, returns the node responsible for the corresponding resource (i.e., data item, computational job, shared network printer, etc.). A search requires traversing $D = O(\log N)$ nodes.

- *P2 – Load Balancing.* A randomly chosen node is responsible for at most a $(1+\delta)/n$-fraction of the key values (and the corresponding resources) for an arbitrarily small $\delta > 0$ depending on sufficiently large n.

- *P3 – Linking Rules.* Each node w links to nodes in a ***neighbor set*** L_w and the rules for forming L_w are known globally. Any node may determine the elements in L_w by performing searches.[9] There are also $O(\text{poly}(\log N))$ nodes whose IDs dictate that w is a neighbor (see the Appendix C). Again, any node may verify this by performing searches. The number of links on which a node is incident is the ***degree*** of w, and every node has the same degree asymptotically.

- *P4 – Congestion Bound.* The ***congestion*** is $C \leq \text{polylog}(N)/N$ where congestion is the maximum probability (over all nodes) that a node is traversed in a search initiated at a randomly chosen node for a randomly chosen point in $[0, 1)$.

Note that G is not robust to bad nodes, but it does provide the underlying topology for our robust construction. For any G satisfying the above properties, our results apply.

We emphasize that many constructions for G exist such as Chord [46], the distance-halving construction in [36], Viceroy [31], Chord++ [4], D2B [18], FISSIONE [28], and Tapestry [51]. Other non-DHT constructions with congestion bounds are also likely suitable input graphs, such as skip graphs [2,3] and hyperrings [6],

[7] For ease of exposition in our proofs, we let β be small (this is standard in the literature) since we are not trying to optimize this quantity. However, larger values of β are likely possible.

[8] We note that the number of nodes N in G may differ from the number of IDs n used in our robust construction.

[9] For example, in Chord [46], the neighbors of w are (1) the successor and predecessor of w, and (2) the successors of the points $w + \Delta(i)$ where $\Delta(i)$ is an exponentially increasing distance in the ID space for integers $i = 1, ..., O(\lg(N))$. Any node may verify that $u \in L_w$ via a search on $w + \Delta(i)$ and checking that the result is u.

where a *membership vector* or *padding sequence* is equivalent to an ID, respectively. In these latter constructions, nodes typically correspond to data items instead of machines; however, one can map data items to a network of nodes that preserves this topology [1].

Node Identifiers and Proof-of-Work. Each node owns an ID which, for simplicity, is a value in $[0, 1)$ (adequate precision is obtained using $O(\log n)$). Important properties that our system guarantees are:

- IDs expire after a period of time that can be set by the system designers.
- A claim to own an ID can be verified by any good node.
- The adversary is limited to roughly βn IDs u.a.r. from $[0, 1)$.

These properties are established via a PoW scheme whereby a node must solve a computational puzzle in order to obtain an ID. Given space constraints and that the bulk of our results are proved without the need to reference these details, we delay their discussion until Section 4.

Throughout this paper, we make the *random oracle assumption* [10]: there exist hash functions, h, such that $h(x)$ is uniformly distributed over h's range, when any x in the domain of h is input to h for the first time. We assume that both the input and output domains are the real numbers between 0 and 1. In practice, h may be a cryptographic hash function, such as SHA-2 [37], with inputs and outputs of sufficiently large bit lengths

We make use of the following well-known concentration results.

Theorem 1. *(Chernoff Bounds [33]) Let $X_1, \ldots, X_N$ be independent indicator random variables such that $\Pr(X_i) = p$ and let $X = \sum_{i=1}^{N} X_i$. For any δ, where $0 < \delta < 1$, the following holds:*
$$\Pr(X > (1+\delta)\, E[X]) \leq e^{-\delta^2\, E[X]/3} \text{ and } \Pr(X < (1-\delta)\, E[X]) \leq e^{-\delta^2\, E[X]/2}$$

Theorem 2. *(Method of Bounded Differences [15]) Let f be a function of the variables $X_1, ..., X_N$ such that for any b, b' it holds that $| f(X_1, ..., X_i = b) - f(X_1, ..., X_i = b') | \leq c_i$ for $i = 1, ..., N$. Then, the following holds:*
$$Pr(f > E[f] + t) \leq e^{-t^2/(2\sum_i c_i^2)} \text{ and } Pr(f < E[f] - t) \leq e^{-t^2/(2\sum_i c_i^2)}$$

1.4 Overview and Our Main Result

As discussed above, reducing quorum size is desirable but gives rise to the possibility of bad quorums. In Section 2, we demonstrate how to achieve $1/\text{poly}(\log n)$-robustness with quorums of size $\Theta(\log \log n)$ when there is no churn. This argument leverages the bound on congestion given by the input graph, along with carefully tallying of the fraction of ID space which cannot be searched.

This result is applied in Section 3 where we show that $1/\text{poly}(\log \log n)$-robustness can be maintained despite churn. A key component of our construction is the use of two graphs (composed of quorums) that, when used in tandem, limit the number of bad quorums that can be formed.

Finally, in Section 4, we describe how PoW is used to provide the guarantees on node IDs discussed in Subsection 1.3. The main challenge is defending against an adversary that wishes to store a large number of IDs for use in a massive future attack (i.e., a pre-computation attack).

Our main result is the following:

Theorem 3. *Assume an input graph G that satisfies $P1$ - $P4$. If the adversary has at most βn computational power, then our construction using $|Q| = O(\log \log n)$ provides the following guarantees w.h.p. over a polynomial number of join and departure events:*

- *all but a $1/\text{poly}(\log n)$-fraction of quorums are good.*
- *all but a $1/\text{poly}(\log n)$-fraction of nodes can successfully search for all but a $1/\text{poly}(\log n)$-fraction of the resources.*

That is, our construction provides $1/\text{poly}(\log \log n)$-robustness. This yields bounds on the cost metrics discussed in Section 1:

- *robust computation incurs* $O(\text{poly}(\log\log n))$ *message complexity.*
- *robust routing incurs* $O(\log n\,\text{poly}(\log\log n))$ *message complexity.*
- *expected* $O(\text{poly}(\log\log n))$ *state maintenance.*

where the cost of state maintenance follows from using a graph G as defined in [18], [31], or [36].

Note that these are substantial improvements over the costs described in Section 1, particularly with respect to robust computation and state maintenance.

Can we do better? A natural question is whether substantially better results are possible and we offer some intuition for why this seems unlikely. With $|Q| = \Theta(\log\log n)$, the probability of a bad quorum is roughly $1/\text{poly}(\log n)$. Given that most search algorithms require (roughly) a logarithmic number of hops, the probability of avoiding any bad quorums along the search path is (roughly) $\sum_1^{\log n} 1/\text{poly}(\log n) < 1$ by a union bound.

Now consider a smaller quorum of size of, say, $\Theta(\log\log\log n)$. The probability of a bad quorum is (roughly) $1/\text{poly}(\log\log n)$ and over a logarithmic number of hops, a union bound fails to offer us a probability less than 1 and routing is likely to fail.

In this sense, our choice of $|Q|$ appears to be pushing the limits of what is possible when designing robust systems with quorums.

2 Almost-Everywhere Routing – The Static Case

We first prove results for the static case since this is a useful building block for the dynamic case.

2.1 The Quorum Graph

Given our input graph G, our approach involves the creation of a ***quorum graph*** $\mathcal{Q}$ which can be viewed as replacing each vertex w in G with a quorum Q_w.

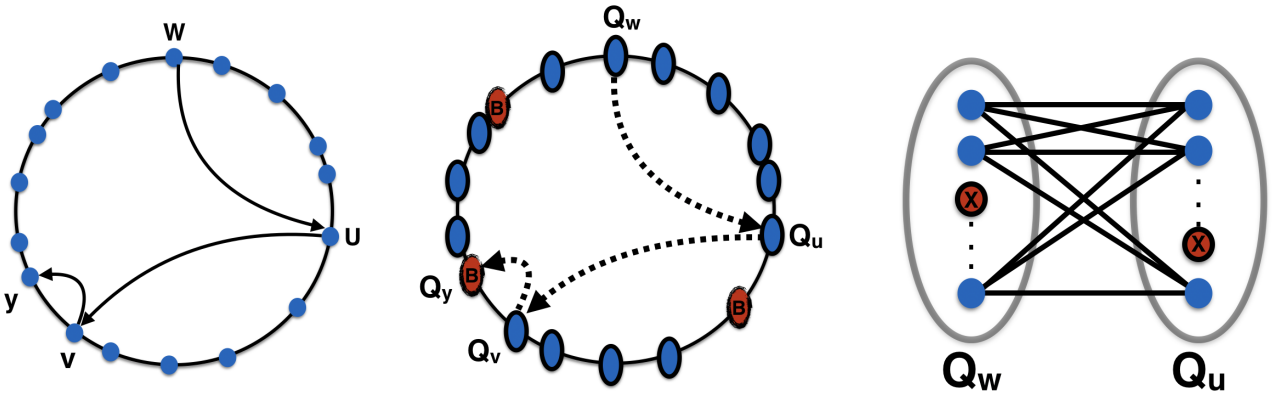
Other aspects are analogous to G. The neighbor set L_w consists of quorums as elements. Edges are directed from quorum Q_w to quorum Q_v – denoted by (Q_w, Q_v) – and signifies that $Q_v \in L_w$. A search in $\mathcal{Q}$ proceeds over these edges as it would in G, except that quorums are being traversed instead of individual nodes. The edge directionality indicates the way in which a search traversing Q_w proceeds; other communication may occur in both directions. The edge (Q_w, Q_v) is realized in the network by all-to-all links between (at least) the good members in Q_w and Q_v.

For ease of presentation, we often speak of quorums being uniformly distributed in the ID space; by this, we mean the leaders of the quorum are uniformly distributed. Congestion is similarly defined for a quorum graph: the probability that a random lookup traverses a quorum (that is, traverses the leader and, by extension, at least the good members of its quorum).

In proving $\mathcal{Q}$ satisfies almost-everywhere routing, we consider the following steps which capture the impact of the adversary:

- *S1.* $\mathcal{Q}$ inherits the properties of the input graph G; each quorum Q_w has $|L_w|$ neighbors, $\text{poly}(\log n)$ degree, and $\mathcal{Q}$ has congestion $C = O(\log^c n/n)$ for a constant $c \geq 0$.
- *S2.* Each quorum is red independently with probability $p_f \leq \frac{1}{\log^k n}$ for a tunable constant $k > 0$.
- *S3.* The adversary adds or deletes edges between red quorums only.

Overview of Analysis. We clarify a few points before presenting our arguments in the next section. A search in $\mathcal{Q} = (V, E)$ is said to ***fail*** if it traverses any red quorum. Pessimistically, we assume the result of a failed search is dictated by the adversary; otherwise, the search ***succeeds***.



(1) (2)

Figure 1: (1) Left: An input graph G with nodes w, u, v, and y. (1) Right: A quorum graph with corresponding quorums Q_w, Q_u, Q_v, and Q_y. Red quorums are marked with a "B". Large dashed arrows represent quorum-to-quorum links. (2) Links between individual nodes associated with members of quorums Q_w and Q_u. The good nodes (colored blue) link to all nodes in the opposite quorum, while Byzantine nodes (colored red) may attempt to establish links arbitrarily.

Informally, blue quorums correspond to good quorums with their neighbors correctly established, while red quorums correspond to bad quorums *or* those quorums with at least one incorrect neighbor. The utility of this coloring scheme will become clearer when we address churn in Section 3.2.

The value of p_f in S2 corresponds to the probability that a quorum is bad or has at least one incorrect neighbor (i.e., is red). To provide intuition, note that if we select $\Theta(\log \log n)$ nodes u.a.r., then the probability that more than a $1/3$-fraction are bad is $O(1/\text{poly}(\log n))$ by a Chernoff bound. A similar bound can be derived on the probability of incorrectly setting up neighbors. Keeping p_f upper bounded by $\frac{1}{\log^k n}$ with churn is non-trivial, and this is argued later in Section 3.2.

Edges in $\mathcal{Q}$ are directed from a quorum to its neighbors and a search traverses these edges as it proceeds. In our analysis, special attention is paid to those edges (Q, Q') such that Q is blue and Q' is red; we call Q' a ***border quorum***. Informally, border quorums can be viewed as forming a boundary point to the "community" of red quorums and a search that encounters a border quorum fails.

Since the adversary controls all red quorums, it is free to insert or delete edges between red quorums; hence the motivation for $S3$. However, edges involving at least one blue quorum are not modified. This corresponds to the fact that the adversary cannot modify the blue quorum's notion of who its neighbors are (since this is kept consistent by the good nodes who are in the majority), although the red quorum may certainly ignore or corrupt incoming messages from that blue quorum.

2.2 Analysis

In $\mathcal{Q}$, a search starting at any quorum Q_i and terminating at the first faulty quorum is a ***route***. Starting at any quorum Q_i, the union of all routes induces a ***search tree***; there is one search tree per quorum.[10]

For a quorum Q_v, we define ***responsibility*** of Q_v to be the sum over all n search trees, $\mathcal{T}$, of the probability that a search using $\mathcal{T}$ for a random point in $[0, 1)$ will traverse Q_v (by this, we mean at least the good quorum members partake in the search); denote this by $\rho(v)$. We are interested in the aggregate responsibility of all red quorums. Note that this is equivalent to examining the aggregate responsibility of all border quorums since a search must traverse a border quorum before traversing any other red quorum.

[10]We note that it is possible that tracing through routes may yield the occasional cycle depending on the distribution of nodes in the ID space; however, this does not impact our analysis.

Lemma 1. *With high probability $\rho(v) = O(\log^c n)$ for each border quorum Q_v.*

Proof. By S1, w.h.p. the search in a random tree for a random point will traverse Q_v with probability $C = O(\log^c n/n)$. □

For a fixed search tree $\mathcal{T}$, we say that a search for a key value ***fails*** if the appropriate route in $\mathcal{T}$ traverses a red quorum. Let X be a random variable that is the sum over all search trees, $\mathcal{T}$, of the probability that a search in $\mathcal{T}$ for a random key value fails. The randomness of X depends on which quorums are red.

Lemma 2. $E(X) \leq O(p_f n \log^c n)$.

Proof. For some fixed quorum Q_v in some fixed search tree, let X_v be a random variable that equals $\rho(v)$ if Q_v is a border quorum, and 0 otherwise. Note that $X \leq \sum_v X_v$. By linearity of expectation, w.h.p. $E(\sum_v X_v) = \sum_v E(X_v) = O(p_f n \log^c n)$ by Lemma 1. □

Lemma 3. $Pr(X \geq (1+\epsilon)p_f n \log^c n) \leq e^{-O(\epsilon^2 p_f^2 n/\log n)}$ *where $\epsilon > 0$ is an arbitrarily small constant depending only on n.*

Proof. For some fixed quorum Q_v in some fixed tree, let X_v be a random variable that equals $\rho(v)$ if Q_v is a border quorum, and 0 otherwise. We will bound $\sum_v X_v$, which is always at least as large as X. Let $f(X_1, ..., X_n) = \sum_v X_v$. By Lemma 1, for any fixed X_i, $|f(..., X_i = x, ...) - f(..., X_i = x', ...)| = O(\log^c n)$. Thus, we can apply Theorem 2 with $c_i^2 = O(\log^{2c} n)$ for all $1 \leq i \leq n$. We have that:

$$Pr(|X - E(X)| \geq \lambda) \leq e^{-\lambda^2/(dn \log^{2c} n)}$$

for some constant $d > 0$. Setting $\lambda = \epsilon p_f n \geq \epsilon n/\log^k n$:

$$Pr(|X - E(X)| \geq \lambda) \leq e^{-\epsilon^2 n/(2\log^{2(c+k)} n)}$$

Plugging in $E(X) = O(p_f n \log^c n)$ from Lemma 2 yields the result. □

Lemma 4. *With probability at least $1 - e^{-O(p_f^2 n/\log^{2(c+k)} n)}$ any search from a random quorum to a random point in $[0, 1)$ succeeds with probability $1 - O(1/\log^{k-c} n)$ where $k \geq c + 1$.*

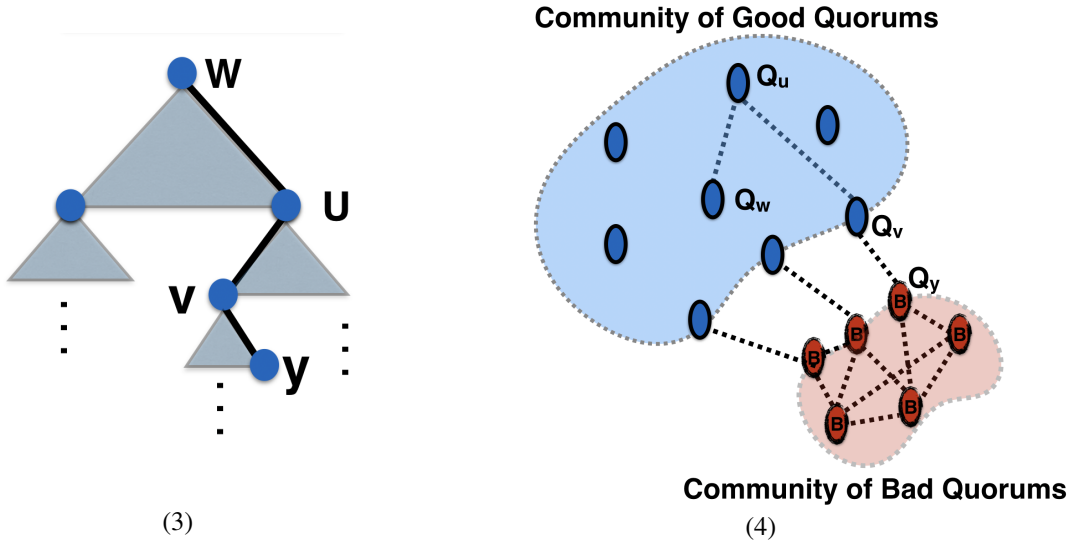
Proof. Fix any $\epsilon > 0$. By Lemma 3, we have $Pr(X \geq (1+\epsilon)p_f n \log^c n) \leq e^{-O(\epsilon^2 p_f^2 n/\log^{2(c+k)} n)}$. We can consider X to be the total space over all search trees that can not be reached because of the red quorums. Hence, if we pick a tree uniformly at random from which to start a search, and search for a random point, then the probability of success is exactly $X/n = O(1/\log^{k-c} n)$ by S2. □

3 Almost-Everywhere Routing - The Dynamic Case

We now consider the case where nodes can join and depart the system. Time is divided into disjoint consecutive windows of T time ***steps*** called ***epochs*** indexed by $j \geq 1$; we discuss the setting of T further in Subection 4.1. In any epoch j, there are:

- two ***old*** quorum graphs $\mathcal{Q}_1^{j-1}$ and $\mathcal{Q}_2^{j-1}$, each with n nodes.
- two ***new*** quorum graphs $\mathcal{Q}_1^j$ and $\mathcal{Q}_2^j$, each with at most n nodes.

We emphasize that the use of two quorum graphs per epoch is critical. A naive approach is to use a *single* quorum graph in the current epoch in order to build a new quorum graph in the next epoch. However, this approach will fail because errors from bad quorums will accumulate over time and we give some intuition for why.



(3) (4)

Figure 2: (3) A sketch of the search tree for node w. (4) A grouping of quorums into good and bad communities. The edge (Q_v, Q_y) bridges the community and any search that traverses this edge from the blue community is assumed to fail. Quorum Q_y is a border quorum.

Informally, in epoch i, we have a process where (1) bad quorums build new bad quorums, and (2) good quorums build bad quorums with some failure probability $p_f^i > 0$ that depends on the current number of bad quorums. Therefore, in the next epoch $i+1$, the population of bad quorums has increased and so has p_f^{i+1}. This increasing error probability will continue until a constant fraction of the quorums are bad (instead of the desired $1/\text{poly}(\log n)$ fraction).

By using two quorum graphs, we can upper bound p_f^i for all i. In particular, we bound this error as $p_f < 1/\frac{1}{\log^k n}$ as this is necessary to invoke our result for the static case (recall S2 in Sectino 2).

The new quorum graphs are built using the old quorum graphs over the n deletions and additions that occur in the current epoch j; we describe this in Subsection 3.1. By the end of epoch j, the old quorum graphs $\mathcal{Q}_1^{j-1}$ and $\mathcal{Q}_2^{j-1}$ are no longer needed, and the new ones $\mathcal{Q}_1^j$ and $\mathcal{Q}_2^j$ are complete.

3.1 Building New Quorum Graphs

We describe how the new quorum graphs $\mathcal{Q}_1^j$ and $\mathcal{Q}_2^j$ are created. Then, in Section 3.2, we prove that w.h.p. this construction preserves almost-everywhere routing.

Preliminaries. Over the T steps of epoch $j-1$, nodes that wish to participate in the system join.[11] These new nodes will be able to use the system in epoch j. Nodes are assumed to know when the system came online (i.e. step 0).[12] Since T is set when the system is designed, any node that wishes to join knows when the current epoch ends and the next one begins.

For now, we assume each good node possesses a single ID which is valid for the T steps of the current epoch. After the epoch comes to an end, each node must obtain a new ID for use in the next epoch. The details of how this is performed and enforced are given later in Section 4; for now, we assume these properties for IDs exist.

[11] As with much of the literature, we do not address concurrency. Since a join or departure event require updating only $\text{poly}(\log n)$ links in a quorum graphs, we assume that there is sufficient time between events to do so.

[12] This is a fixed parameter included as part of the application, along with T, the hash functions, and various constants.

Each good node v maintains the same ID in $\mathcal{Q}_1^{j-1}$ and $\mathcal{Q}_2^{j-1}$. Recall from Section 1.3 that IDs expire after a tunable period of time. Prior to this expiration, Q_v is said to be ***active*** and it can initiate searches. When v's ID expires, its quorum Q_v (this includes v) should remain in both old graphs for an additional T steps. During these steps, Q_v will forward communications but it cannot initiate searches; we say that Q_v is ***passive***.

A **new node w** with a random ID is bootstrapped into the new quorum graph by a ***bootstrapping quorum*** denoted by Q_{boot}. Throughout, we assume that a joining node knows a good bootstrapping quorum; we discuss this further in the Appendix D.

Making a Quorum-Member Request. In $\mathcal{Q}_1^{j-1}$, the i^{th} member of Q_w is the successor to $h_1(w,i)$ for $i = 1, ..., d_2 \ln \ln n$ where h_1 is a secure hash function (and d_2 is defined with respect to quorum size in Section 1.3). In *both* $\mathcal{Q}_1^{j-1}$ and $\mathcal{Q}_2^{j-1}$, a search for each successor of $h_1(w,i)$ is performed by the bootstrapping quorum and suc($h_1(w,i)$) is solicited for membership in Q_w.

A similar process occurs to form the quorums for Q_w in $\mathcal{Q}_2^{j-1}$, except that a different secure hash function h_2 is used. Therefore, the membership of Q_w is likely different in each quorum graph.

Making a Neighbor Request. If w and u are neighbors in the input graph, then Q_w and Q_u should be neighbors in the quorum graph (recall that this entails all-to-all links between the members of both quorums). By property P3 of the input graph G, each node $u \in L_w$ is dictated by w's ID. On behalf of w, Q_{boot} performs a search to locate each such neighbor u. In this way, Q_{boot} allows u (and Q_u) to learn about w and agree to set up a link in the respective quorum graph.

Verifying Requests. The adversary may attempt to have many good nodes join as neighbors or members of a bad quorum. This attack is problematic since good nodes have their resources consumed by maintaining too many neighbors or joining too many quorums; that is, this attack increases the state cost (see Section 1). Therefore, any such request must be verified:

Verifying a Quorum-Membership Request. When node u is asked to become a member of quorum Q_w, node u first checks whether $h_1(w,i)$ for the appropriate i (that accompanies the request) has a value which is within $[u - (c' \ln n)/n, u)$ for a constant $c' > 0$ sufficiently large (we relax notation such that u refers to the name of the node as well as its ID value). If $h_1(w,i)$ does not fall within this distance, then u immediately rejects the request.

Else, u verifies the request by performing a search on $h_1(w,i)$. If this returns u, then the request is considered verified and u becomes a neighbor of w; otherwise, the request is rejected. The correctness of the quorum construction is argued in Lemma 7 and the bound on state cost maintained by this verification procedure is given in Lemma 10.

Verifying a Neighbor Request. A node u that is asked to become a neighbor of node w (and thus establish links between the members of Q_u and Q_w) must verify the request. Recall that by property P3 of the input graph, u can determine independently by a search whether u should indeed be a neighbor of w. If this search returns u, then the request is verified and u becomes a neighbor of w; otherwise, the request is rejected. The correctness of the resulting neighbor set is argued in Lemma 8 and the bound on state cost for this verification procedure is given in Lemma 10.

Performing a Search. Throughout epoch j, each new node w performs searches only using the old quorum graphs $\mathcal{Q}_1^{j-1}$ and $\mathcal{Q}_2^{j-1}$. This is done by forwarding the request to Q_{boot} and forwarding on the search from that position. Since Q_{boot} was active when w joined, Q_{boot} will remain in the system – perhaps in a passive state –to facilitate searches in the old quorum graphs for another T steps. After this point, $\mathcal{Q}_1^j$ and $\mathcal{Q}_2^j$ are complete and w may issue its own searches using these quorum graphs.

Why must w forward its request through Q_{boot}? Over the duration of epoch j, the new quorum graphs are still under construction. In an extreme case, for example, w might be the first node to join $\mathcal{Q}_1^j$ and $\mathcal{Q}_2^j$.

Note that, after each join event when another new node is bootstrapped into the two new quorum graphs, Q_w may need to update its neighbor links $\mathcal{Q}_1^j$ and $\mathcal{Q}_2^j$ and this is done via searches in the old quorum graphs $\mathcal{Q}_1^{j-1}$ and $\mathcal{Q}_2^{j-1}$. This is also true if a new node decides to depart a new quorum graph (even before its completed).

Once epoch $j+1$ starts, the new quorum graphs $\mathcal{Q}_1^j$ and $\mathcal{Q}_2^j$ are to be used. At this point, Q_w will initiate any search using its own links in these graphs (rather than relying on Q_{boot} which may no longer be present in the system).

3.2 Analysis

In this section, we prove that old quorum graphs satisfying $S1$, $S2$ and $S3$ can be used to construct new quorum graphs that preserve $S1$, $S2$, and $S3$. Due to space constraints, some proofs are provided in the Appendix.

Properties P1-P4 of input graph G are critical to our arguments. However, a prerequisite to these properties is that *all* IDs are selected uniformly at random (see Section 1.3) which is untrue if the adversary chooses to add only some of its bad nodes; for example, maybe only bad nodes with IDs in $[0, \frac{1}{2})$ are added by the adversary. Intuitively, this should not interfere with any of the properties.

In the following, we may consider G' to be a modified input graph which uses the same construction as G, but is subject to an adversary that only includes a subset of its nodes (from a larger set of nodes with u.a.r. IDs).

Lemma 5. *Consider a graph G' where the nodes are formed from two sets:*

- *$\mathcal{N}_1$ consists of at least $(1-\beta)n$ nodes with IDs selected u.a.r. from $[0,1)$.*
- *$\mathcal{N}_2$ is an arbitrary subset of at most βn nodes with IDs selected u.a.r. from $[0,1)$.*

W.h.p., under the same construction as the input graph G, graph G' has properties P1 - P4.

Throughout, the above result is assumed — that properties P1-P4 continue to hold if the adversary includes only a subset of its IDs — even if we do not always make it explicit (for example, P1 is used throughout, P2 is used in Lemma 6, P4 in Lemma 9, and P3 in Lemma 10).

As described above, a node u performs searches on random key values (via hashing under the random oracle assumption) in order to locate members for a new quorum Q. But if that key value maps to a bad node, then this results in a bad member being added to the quorum. We can bound the probability of this event:

Lemma 6. *W.h.p. a random key value in an old quorum graph maps to a bad node with probability at most $(1+\delta')\beta$ for an arbitrarily small constant $\delta' > 0$ depending only on sufficiently large n.*

In the following, let $q_f = O(1/\log^{k-c} n)$ be the probability that a search for a random key in an old quorum graph $\mathcal{Q}_i^{j-1}$ fails; this is dictated by Lemma 4. Recall that a quorum Q is ***good*** if $d_1 \ln\ln n \leq |Q| \leq d_2 \ln\ln n$ members, for constants $d_1 < d_2$, and at most $|Q|/3$ members are bad.

Lemma 7. *A new quorum is bad with probability at most $q_f^2 d_2 \ln\ln n + 1/\log^{d'} n$ for a tunable constant $d' > 0$ depending on d_2.*

Proof. For a new node w, there are two ways in which building Q_w may fail. First, a search for a quorum member may fail (i.e., encounters a bad quorum). Given a point $h_1(w,i)$, the probability that both searches in $\mathcal{Q}_1$ and $\mathcal{Q}_2$ fail is at most q_f^2. By a union bound, the probability of such a dual failure occurring over $d_2 \ln\ln n$ searchers is at most $q_f^2\, d_2 \ln\ln n$.

Second, the search succeeds but returns $\text{suc}(h(w,i))$ where $\text{suc}(h(w,i))$ is a bad node (even though its quorum has a good majority). Since $h(w,i)$ is a random point (under the random oracle assumption), this event occurs with probability at most $(1+\delta)\beta$ by Lemma 6 for an arbitrarily small constant $\epsilon > 0$ given sufficiently large n.

Over $d_2 \ln\ln n$ searches, the expected number of such events is at most $(1+\delta)\beta d_2 \ln\ln n$. The probability of exceeding this expectation by more than a small constant factor is $1/\log^{d'} n$ by a Chernoff bound where the constant $d' > 0$ is tunable depending only on sufficiently large d_2. □

A quorum Q_w should link to all quorums with leaders in the neighbor set L_w as dictated by the input graph G. If Q_w (1) links to any quorum whose leader is not in L_w, or (2) fails to link to any quorum whose leader is in L_w, then Q_w is said to be ***confused***. We now bound the probability of a confused quorum being created.

Lemma 8. *Each quorum in a new quorum graph is confused independently with probability at most q_f^2.*

Proof. Since the bootstrapping quorum is good, the only ways in which a quorum is confused is if the two searches for a neighbor point in the *old* quorum graph $\mathcal{Q}'_i$ both fail. Applying Lemma 4, this occurs with probability at most q_f^2. □

We are now ready to prove that w.h.p. each new quorum graph is almost-everywhere routable.

Lemma 9. *Assume that the adversary adds at most $(1+\epsilon)\beta n$ nodes with u.a.r. IDs to a new quorum graph for an arbitrarily small constant $\epsilon > 0$ depending only on sufficiently large n. Then, w.h.p., each new quorum graph is almost-everywhere routable.*

Proof. To prove this result, we demonstrate equivalence between the construction of a new quorum graph and steps S1, S2, and S3 in Section 2. Recall the terminology in Section 2 and designate all bad quorums and confused quorums as *red*, and all other quorums as *blue*.

Equivalence to S1. By assumption, the adversary has at most $(1+\epsilon)\beta n$ u.a.r IDs. The good nodes also have u.a.r. IDs. Using *all* of these IDs would give congestion C corresponding to the input graph G and thus step S1 would be satisfied. However, the adversary may choose to employ only a subset of its IDs – how does this affect the congestion? By Lemma 5, the resulting congestion is $O(C)$ and we have equivalence with S1 up to a constant factor (which does not affect the argument in Lemma 4).[13]

Equivalence to S2. Satisfying S2 requires enforcing that for each construction of a new quorum graph, the probability of a red quorum is at most $p_f \leq 1/\log^k n$ for a tunable constant $k > 0$. By Lemmas 7 and 8, each quorum is red independently with probability at most:

$$\begin{aligned} &\leq q_f^2 + q_f^2 d_2 \log\log n + \left(1/\log^{d'} n\right) \\ &\leq O\left(\frac{\log\log n}{\log^{2(k-c)} n} + \frac{1}{\log^{d'} n}\right) \\ &\leq p_f \end{aligned}$$

The last line follows by setting d_2 to be sufficiently large such that d' exceeds $2(k-c)$; note d_2 is fixed at the beginning and never needs to be changed throughout the lifetime of the network. Then, setting $k > 2c$ to be a sufficiently large constant yields the necessary inequality with p_f.

Equivalence to S3. The incorrect link structure of confused quorums corresponds to step S3.

Finally, Lemma 4 applies to the new quorum graph and competes the proof. □

We now prove bounds on the number links a good node needs to maintain due to (1) membership in quorums, and (2) being a neighbor of a quorum. This is done by analyzing the verification process described in Section 3.1.

Lemma 10. *In expectation, each good node w in a quorum graph is a member of $O(\log\log n)$ quorums and maintains state on $O(|L_w|)$ quorums that are either neighbors or have w as a neighbor.*

[13] Note that the degraded congestion is not cumulative over the sequence of new quorum graphs. Rather, each new quorum graph with starts with congestion C which can be worsened to $O(C)$ in each instance.

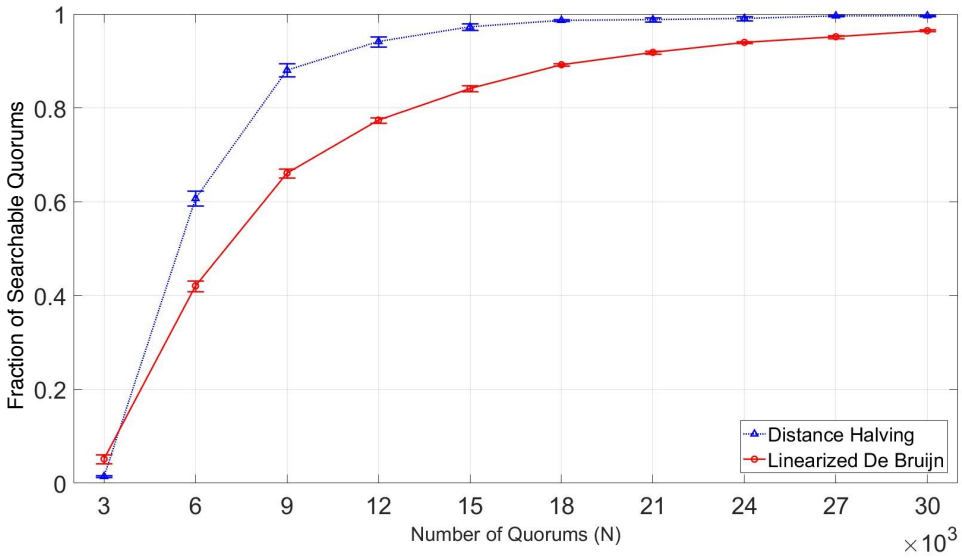


Figure 3: Evaluation of almost-everywhere routing in the Distance-Halving Construction [36] and Linearized De Bruijn network [38].

We can now prove Theorem 3:

Proof. By Lemma 7, all but a $1/\text{poly}(\log n)$-fraction of quorums are good. It follows that for each good quorum, executing a BA or secure multiparty computation schemes that incur $\text{poly}(|Q|)$ message complexity (discussed in Section 1) have $\text{poly}(\log \log n)$ message complexity.

By Lemma 9, all but a $1/\text{polylog}\, n$-fraction of nodes can successfully search for all but a $1/\text{polylog}\, n$-fraction of the resources.

Given that robust routing proceeds all-to-all between quorums and that searches have maximum length $D = O(\log n)$ (recall P1 in Section 1.3), the message complexity is $O(\log n\, (\log \log n)^2)$.

To bound the state cost, we invoke Lemma 10. Each good node w belongs to $O(\log \log n)$ quorums in expectation which implies $O((\log \log n)^2)$ expected state cost. Additionally, in terms of neighbors, w has links (to or from) $O(\text{poly}(\log n))$ quorums. The constructions for G defined in [36], [18], or [31] provide the properties P1-P4, but provide a better bound of $O(1)$ expected degree. Using any such construction allows for a state cost of $O(\log \log n)$ in expectation. Therefore, the total state cost is $O((\log \log n)^2) + O(\log \log n) = O((\log \log n)^2)$. □

3.3 Experiments on Almost-Everywhere Routing

To investigate the effectiveness of our almost-everywhere routing guarantee, we simulated two DHT constructions: The Distance-Halving (DH) construction [36] and the Linearized De-Bruijn (LDB) network [38]. The DH construction is interesting because it offers $O(1)$ expected degree, $O(\log n)$ search latency, and has w.h.p. expected congestion $O(\log^2 n/n)$ (see [4]). The LDB network does not (to our knowledge) have a congestion bound, but it does offer $O(1)$ degree (not just in expectation) in addition to a $O(\log n)$ search latency. This bound on degree is useful since it means that all quorums then possess $\text{poly}(\log \log n)$ state overall in the

quorum graph, whereas with the DH construction, a small number of quorums will have $\Theta(\log n)$ degree and thus have poly$(\log n)$ state.

The quorums in our simulations are programmed to be bad with probability $1/(c \ln n)^k$, where $c = 13/100$ and $k = 28$. This yields a probability ≈ 0.33 for $n = 3,000$ and a probability ≈ 0.0003 for $n = 30,000$. These parameters c and k were chosen to clearly illustrate the effect of n on our almost-everywhere routing guarantee.

We measure the fraction of quorums in the system that are routable by a search operation from a single source quorum; that is, no bad quorum is encountered on the search path from source to destination. For each construction, we tested 15 DHTs per n value and a sample size of 15 source quorums per DHT was used.

The experimental results are plotted in Figure 3, along with error analysis using a 95% confidence interval. The simulation was implemented with the Java programming language, and the plots were generated using MATLAB. The experiment was executed on an Acer Aspire F5-573G laptop with a Windows 10 Operating System, Intel Core i5 CPU, 64-bit Operating System, 8GB RAM, with a 2.30GHz processor.

For the DH construction, the fraction of routable quorums at $n = 3,000$ quorums was 0.0144 while the fraction of routable quorums at $n = 30,000$ quorums was 0.9964. Therefore, as n increases, the almost-everywhere routing property improves dramatically. Similarly, for the LDB network, the fraction of routable quorums at $n = 3000$ quorums was 0.0516 while the fraction of routable quorums at $n = 30,000$ quorums was 0.9646.

4 Computational Puzzles

Up to this point, we have assumed that the adversary can inject into each new quorum graph at most $(1+\epsilon)\beta n$ bad nodes with u.a.r. IDs, and that these IDs can be verified and forced to expire after a period of time (Section 1.3). Given space constraints, we limit our discussion to the main ideas of how to use computation puzzles to guarantee these properties.

4.1 Generating an ID

All nodes are assumed to know two secure hash functions, f and g, with range and domain $[0, 1)$ and that both hash functions satisfy the random oracle assumption.

In the current epoch i, node w is assumed to possess a "globally-known" random string r_{i-1} of $\ell \ln n$ bits. By "globally-known", we mean known to all good nodes except the $1/\text{poly}(\log n)$-fraction from our earlier analysis. We motivate r_{i-1} and describe how it is generated in Subsection 4.2.

Starting at step $T/2$ in the current epoch, each good node begins generating a new ID for use in the next epoch (see Subsection 4.2) as described below.

Generating an ID. To generate an ID, a good node w selects a value σ_w of $\ell \ln n$ random bits (matching the length of r_{i-1}). Then, w XORs these two strings to get $\sigma_w \oplus r_{i-1}$, and checks if $g(\sigma_w \oplus r_{i-1}) \leq \tau$. We assume the value τ is set small enough such that w.h.p. a node requires $(1 \pm \epsilon)T/2$ time steps to find a σ_w that satisfies this inequality, where $\epsilon > 0$ is a tunable (small) positive constant and $T > 0$ is a parameter set when the system is initialized.

We remark that the use of two secure hash functions is not immediately apparent. We justify this design choice below in Lemma 11 when we prove a bound on the adversary's ability to generate IDs.

Observe that T is roughly the maximum session time for any node since its ID is valid for this number of steps; after this, a node must obtain a new ID (although the node, along with its quorum, may remain in a passive state for an additional T steps using this ID). However, T can be set to a large value so that this forced churn is spread out over long periods of time to avoid negatively impacting performance. Given that the designers can estimate the rate of churn for their application and set a (loose) upper bound on n, then they can set T accordingly.

Finally, we note that the adversary may attempt to generate IDs throughout the entire epoch rather than stopping after $T/2$ steps. At worst, the adversary may then actually be in possession of roughly $2\,\beta n$ IDs.

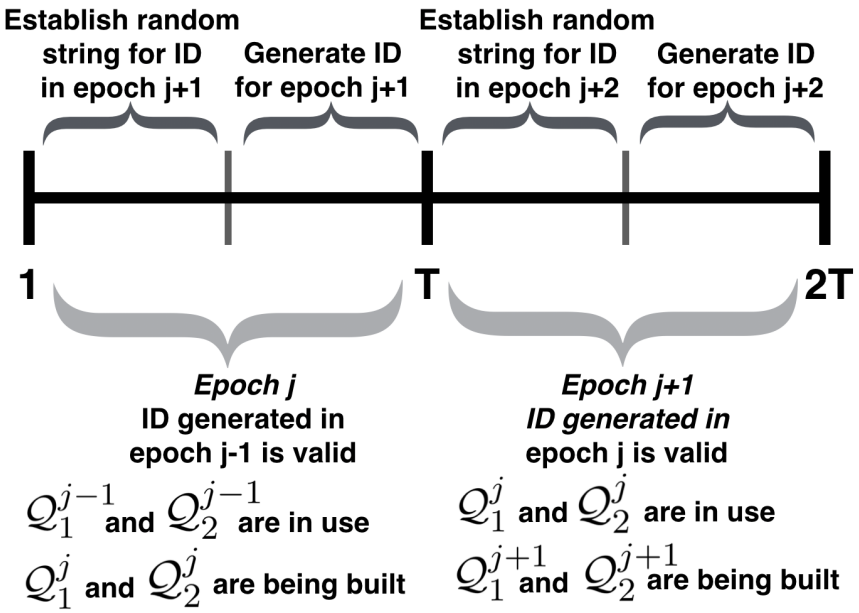


Figure 4: Depiction of the timeline for epochs j and $j+1$. The first $T/2$ steps are dedicated to generating a random string which is used for input into the ID generation step; these details are described later in Section 4. Given this random string, the generation of an ID takes place over the second $T/2$ steps of each epoch. Each ID is valid over the T time steps of the succeeding epoch.

We can address this problem by revising the value of β such that $\beta < 1/8$ which then allows us to recapture the bound that less than $n/4$ IDs in the system are bad (recall from Section 1.3 that we are not optimizing β, and we only care about tolerating an adversary with $\Theta(n)$ computational power) and so we omit further discussion of this issue.

Why Use Two Hash Functions? Consider using a single secure hash function f to assign IDs; that is, if $f(x) < \tau$, then x is a valid ID. In this case, the adversary has a strategy where it may restrict itself to small inputs x in order to confine its solutions to yielding small IDs. In other words, the IDs obtained by the adversary will not be u.a.r. from $[0, 1)$. This motivates the use of composing two secure hash functions, f and g, as described above. We prove the following:

Lemma 11. *W.h.p., the adversary can generate at most $(1+\epsilon)\beta n$ IDs per epoch and these IDs are uniform at random in $[0, 1)$.*

Proof. Since the adversary has βn computational power to expend over this epoch, w.h.p. it can generate at most $(1+\epsilon)\beta n$ solutions σ_v such that $g(\sigma_w \oplus r_{i-1}) \leq \tau$ within an epoch where the constant $\epsilon > 0$ is can be made arbitrarily small depending only on sufficiently large n. By the random oracle assumption, applying f to these solutions yields at most $(1+\epsilon)\beta n$ IDs. □

ID Verification. Upon receiving a message from some node w, a good node u ***verifies*** w's ID. This could be done naively by having w send σ_w to u who checks that $g(\sigma_w \oplus r_{i-1}) \leq \tau$ and that $f(g(\sigma_w \oplus r_{i-1}))$ evaluates correctly to the claimed ID (note that u already has r_i since it is globally-known). Unfortunately, this allows u to steal σ_w if u is bad.

To avoid this issue, we assume a zero-knowledge scheme for revealing the pre-image of the hashing; such a scheme is provided for the SHA family [22]. This allows w to prove the validity of σ_w without revealing it.

If w's ID cannot be verified, then u simply ignores w going forward. Note that w's current ID will not be valid in the next epoch since it is signed by the older string r_{i-1} (rather than the next globally-known random

string r_i); that is, w's ID will have expired. Nodes with IDs that are not verified are effectively removed from the system; that is, they may consort with bad nodes, but they have no interactions with good nodes.

4.2 Generating Global Random Strings

Imagine if no random string was used in the creation of IDs described above in Subsection 4.1. The adversary would know the format of the ID-generation puzzles, and so could spend time computing a large number of IDs, and then use these IDs all at once to overwhelm the system at some future point. This is a ***pre-computation attack***.

Signing IDs with a random string prevents such an attack as it is impossible for the adversary to know far in advance how to generate identifiers. We provide a scheme where random strings are generated and propagated throughout the network in order to be used in ID generation and, consequently, safeguard against a pre-computation attack.

Due to space constraints, this content is provided in Appendix B. Our main result is:

Lemma 12. *W.h.p., the protocol for propagating strings (i) guarantees that, for each good node w, its string used for generating an ID is known is to each good node, (ii) the number of strings stored by each node is $O(\ln n)$, and (iii) has message complexity $\tilde{O}(n \ln T)$.*

We note that, averaged out over the epoch, this message complexity is low.

5 Conclusion and Future Work

We have argued that quorums of size $O(\log \log n)$ can be used to tolerate a powerful adversary that controls a constant fraction of the computing resources in the network. Intuition is also provided as to why this seems to be (roughly) the smallest quorum size that leads to a useful robustness result.

A portion of our result relies on PoW techniques to limit the number of IDs the adversary controls. While it is interesting that these techniques can be leveraged to greatly reduce quorum size – yielding the consequent savings in message complexity and state maintenance – an open question is whether the computational costs for the PoW component can be reduced. Might there be a way to avoid the constant solving of puzzles? Is there an approach that would only utilize puzzle solving when the system is determined to be under attack?

References

[1] James Aspnes, Jonathan Kirsch, and Arvind Krishnamurthy. Load Balancing and Locality in Range-Queriable Data Structures. In *Proceedings of the Symposium on Principles of Distributed Computing (PODC)*, pages 115–124, 2004.

[2] James Aspnes and Gauri Shah. Skip Graphs. In *Proceedings of the* 14^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 384–393, 2003.

[3] James Aspnes and Udi Wieder. The Expansion and Mixing Time of Skip Graphs. In *Proceedings of the Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 126–134, 2005.

[4] Baruch Awerbuch and Christian Scheideler. Chord++: Low-Congestion Routing in Chord. Available at: `www.cs.jhu.edu/~scheideler/papers/chord4.ps.gz`, 2004.

[5] Baruch Awerbuch and Christian Scheideler. Group Spreading: A Protocol for Provably Secure Distributed Name Service. In *Proceedings of the* 31^{st} *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 183–195, 2004.

[6] Baruch Awerbuch and Christian Scheideler. The Hyperring: a Low-Congestion Deterministic Data Structure for Distributed Environments. In *Proceedings of the* 15^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 318–327, 2004.

[7] Baruch Awerbuch and Christian Scheideler. Robust Random Number Generation for Peer-to-Peer Systems. In *Proceedings of the* 10^{th} *International Conference On Principles of Distributed Systems (OPODIS)*, pages 275–289, 2006.

[8] Baruch Awerbuch and Christian Scheideler. Towards a Scalable and Robust DHT. In *Proc. of the* 18^{th} *ACM Symposium on Parallelism in Algorithms and Architectures*, pages 318–327, 2006.

[9] Baruch Awerbuch and Christian Scheideler. Towards Scalable and Robust Overlay Networks. In *Proc. of the* 6^{th} *International Workshop on Peer-to-Peer Systems (IPTPS)*, 2007.

[10] Mihir Bellare and Phillip Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.

[11] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S. Wallach. Secure Routing for Structured Peer-to-Peer Overlay Networks. In *Proceedings of the* 5^{th} *Usenix Symposium on Operating Systems Design and Implementation (OSDI)*, pages 299–314, 2002.

[12] Varsha Dani, Valerie King, Mahnush Movahedi, and Jared Saia. Quorums Quicken Queries: Efficient Asynchronous Secure Multiparty Computation. In *International Conference on Distributed Computing and Networking*, pages 242–256. Springer, 2014.

[13] Mayur Datar. Butterflies and Peer-to-Peer Networks. In *Proceedings of the 10th Annual European Symposium on Algorithms (ESA)*, pages 310–322, 2002.

[14] John Douceur. The Sybil Attack. In *Proceedings of the Second Internation Peer-to-Peer Symposium (IPTPS)*, pages 251–260, 2002.

[15] Devdatt Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 1st edition, 2009.

[16] Amos Fiat and Jared Saia. Censorship Resistant Peer-to-Peer Content Addressable Networks. In *Proceedings of the* 13^{th} *ACM Symposium on Discrete Algorithms (SODA)*, pages 94–103, 2002.

[17] Amos Fiat, Jared Saia, and Maxwell Young. Making Chord Robust to Byzantine Attacks. In *Proceedings of the* 13^{th} *European Symposium on Algorithms (ESA)*, pages 803–814, 2005.

[18] Pierre Fraigniaud and Philippe Gauron. D2B: A De Bruijn Based Content-addressable Network. *Theor. Comput. Sci.*, 355(1):65–79, April 2006.

[19] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The Bitcoin Backbone Protocol: Analysis and Applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–310. Springer, 2015.

[20] Rachid Guerraoui, Florian Huc, and Anne-Marie Kermarrec. Highly Dynamic Distributed Computing with Byzantine Failures. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 176–183, 2013.

[21] Kristen Hildrum and John Kubiatowicz. Asymptotically Efficient Approaches to Fault-Tolerance in Peer-to-Peer Networks. In *Proceedings of the* 17^{th} *International Symposium on Distributed Computing*, pages 321–336, 2004.

[22] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge Using Garbled Circuits: How to Prove Non-algebraic Statements Efficiently. In *Proceedings of the 2013 ACM Conference on Computer Communications Security (CCS)*, pages 955–966, 2013.

[23] Apu Kapadia and Nikos Triandopoulos. Halo: High-Assurance Locate for Distributed Hash Tables. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2008.

[24] Bruce Kapron, David Kempe, Valerie King, Jared Saia, and Vishal Sanwalani. Fast Asynchronous Byzantine Agreement and Leader Election with Full Information. In *Symposium on Discrete Algorithms (SODA)*, pages 1038–1047, 2008.

[25] Valerie King and Jared Saia. Breaking the $O(n^2)$ Bit Barrier: Scalable Byzantine Agreement with an Adaptive Adversary. In *Proceedings of the* 29^{th} *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 420–429, 2010.

[26] Valerie King and Jared Saia. Breaking the $O(n^2)$ Bit Barrier: Scalable Byzantine Agreement with an Adaptive Adversary. *Journal of the ACM (JACM)*, 58(4), 2011.

[27] Jeffrey Knockel, George Saad, and Jared Saia. Self-Healing of Byzantine Faults. In *Proceedings of the International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 98–112, 2013.

[28] Dongsheng Li, Xicheng Lu, and Jie Wu. FISSIONE: A Scalable Constant Degree and Low Congestion DHT Scheme Based on Kautz Graphs. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 3, pages 1677–1688, 2005.

[29] Frank Li, Prateek Mittal, Matthew Caesar, and Nikita Borisov. SybilControl: Practical Sybil Defense with Computational Puzzles. arXiv:1201.2657, 2012.

[30] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A Secure Sharding Protocol For Open Blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 17–30, 2016.

[31] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. In *Proceedings of the* 21^{st} *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 183–192, 2002.

[32] Silvio Micali. ALGORAND: The Efficient and Democratic Ledger. *CoRR*, abs/1607.01341, 2016.

[33] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge Univ. Press, 1995.

[34] A. Nambiar and M. Wright. Salsa: A Structured Approach to Large-Scale Anonymity. In *Proc. of the* 13^{th} *ACM Conference on Computer and Communications Security*, pages 17–26, 2006.

[35] Moni Naor and Udi Wieder. A Simple Fault Tolerant Distributed Hash Table. In *Proceedings of the Second International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 88–97, 2003.

[36] Moni Naor and Udi Wieder. Novel architectures for P2P applications: the continuous-discrete approach. In *Fifteenth ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 50–59, 2003.

[37] National Institute of Standards and Technology. Secure Hash Standard (SHS). `http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf`, 2015.

[38] Andréa Richa, Christian Scheideler, and Phillip Stevens. Self-stabilizing De Bruijn Networks. In *Proceedings of the 13th International Conference on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 416–430, 2011.

[39] Rodrigo Rodrigues and Barbara Liskov. Rosebud: A Scalable Byzantine-Fault-Tolerant Storage Architecture. Technical Report TR/932, MIT LCS, December 2003.

[40] Rodrigo Rodrigues, Barbara Liskov, and Liuba Shrira. The design of a robust peer-to-peer system. In 10^{th} *workshop on ACM SIGOPS European workshop*, page 2002, 117-124.

[41] George Saad and Jared Saia. Self-Healing Computation. In *Proceedings of the International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 195–210, 2014.

[42] Jared Saia, Amos Fiat, Steve Gribble, Anna Karlin, and Stefan Saroiu. Dynamically Fault-Tolerant Content Addressable Networks. In *Proceedings of the First International Workshop on Peer-to-Peer Systems*, Cambridge, MA, 2002.

[43] Jared Saia and Maxwell Young. Reducing Communication Costs in Robust Peer-to-Peer Networks. *Information Processing Letters*, 106(4):152–158, 2008.

[44] Christian Scheideler and Stefan Schmid. *A Distributed and Oblivious Heap*, pages 571–582. 2009.

[45] Siddhartha Sen and Michael J. Freedman. Commensal Cuckoo: Secure Group Partitioning for Large-Scale Services. *ACM SIGOPS Operating Systems*, 46(1):33–39, February 2012.

[46] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 149–160, 2001.

[47] Peng Wang, Ivan Osipkov, and Yongdae Kim. Myrmic: Secure and Robust DHT Routing. Technical report, University of Minnesota, 2007.

[48] Maxwell Young. Making Chord Robust to Byzantine Attacks. Master's thesis, University of New Mexico, 2006.

[49] Maxwell Young, Aniket Kate, Ian Goldberg, and Martin Karsten. Towards Practical Communication in Byzantine-Resistant DHTs. *IEEE/ACM Transactions on Networking*, 21(1):190–203, February 2013.

[50] Haifeng Yu. Sybil Defenses via Social Networks: A Tutorial and Survey. *SIGACT News*, 42(3):80–101, October 2011.

[51] Ben Y. Zhao, John Kubiatowicz, and Anthony D. Joseph. Tapestry: An Infrastructure for Fault-Resilient Wide-Area Location and Routing. Technical Report UCB//CSD-01-1141, University of California at Berkeley Technical Report, April 2001.

Appendix

A Proofs for Section 3.2

Lemma 5. *Consider a graph G' where the nodes are formed from two sets:*

- *$\mathcal{N}_1$ consists of at least $(1-\beta)n$ nodes with IDs selected u.a.r. from $[0,1)$.*
- *$\mathcal{N}_2$ is an arbitrary subset of at most βn nodes with IDs selected u.a.r. from $[0,1)$.*

W.h.p., under the same construction as the input graph G, graph G' has properties P1 - P4.

Proof. View the ID space as a unit ring, and place on it the nodes from $\mathcal{N}_1$ and $\mathcal{N}_2$. Let the total number of nodes be m where $m \geq (1-\beta)n$.

Moving clockwise from any node, consider a contiguous interval of length $(\lambda \ln m)/m$ where $\lambda > 0$ is any constant. Since IDs in $\mathcal{N}_2$ are selected from a larger set of IDs u.a.r in $[0, 1)$, intuitively the adversary's choice of $\mathcal{N}_2$ cannot significantly change the density/sparseness of nodes on the ring. By Chernoff bounds, regardless of how $\mathcal{N}_2$ is selected, and for any $\lambda > 0$, the following holds:

- with probability at least $1 - (m)^{-\lambda/12}$, every interval contains at least $(\lambda/2) \ln m$ nodes, and
- with probability at least $1 - m^{-\lambda/12}$, every interval contains at most $(3\lambda/2) \ln m$ nodes.

A placement of m nodes on the ring that satisfies these two properties is a ***λ-well-spread placement***. Observe that no matter how the adversary chooses $\mathcal{N}_2$, w.h.p. the adversary's influence on the distribution of IDs is characterized by some λ-well-spread placement. In other words, we can ignore the adversary since the issue now reduces to: what is the probability of a λ-well-spread placement that degrades a property in G'?

We argue by contradiction as follows. Recall the guarantee that the input graph G has some property P with probability at least $1 - 1/m^c$ for a *tunable* constant $c > 0$ (Section 2). Now assume there exists a set $\mathcal{S}$ of λ-well-spread placements that violates this property P for G', and that these occur with aggregate probability $1/m^d > 1/m^c$ for some positive constant d. But this yields a contradiction since, with probability at least $1/m^d > 1/m^c$, placing m nodes u.a.r. on the ring would yield a placement from $\mathcal{S}$ for the input graph G and, therefore, violate the guarantee of property P for G. □

Lemma 6. *W.h.p. a random key value in an old quorum graph maps to a bad node with probability at most $(1 + \delta')\beta$ for an arbitrarily small constant $\delta' > 0$ depending only on sufficiently large n.*

Proof. By property P2 of the input graph G, w.h.p. a randomly chosen node in G is responsible for at most a $(1 + \delta)/n$-fraction of the key values for an arbitrarily small $\delta > 0$ depending on sufficiently large n (and, by Lemma 5, this holds even if the adversary does not add all of its bad IDs). Since the IDs of the adversary are u.a.r., the βn bad nodes are responsible for at most a $(1 + \delta)\beta$-fraction of the key values. □

Lemma 10. *In expectation, each good node w in a quorum graph is a member of $O(\log \log n)$ quorums and maintains state on $O(|L_w|)$ quorums that are either neighbors or have w as a neighbor.*

Proof. We perform our analysis with respect to a good node w. First, we analyze the state cost incurred by w due to quorum-membership requests. Second, we analyze the state cost incurred by w due to (i) neighbors w links *to*, and (ii) those neighbor requests w receives.

State Cost of Quorum-Membership Requests. In the absence of adversarial interference, the number of quorum-member requests that w receives is $O(\log \log n)$ given that such requests are distributed among all nodes u.a.r. and each quorum requires $O(\log \log n)$ members.

Now, we consider the impact of the adversary on this state cost. Consider node w receives a request to join some quorum Q_v. As described in Section 3.1, node w first checks that $h_1(w, i)$ for the appropriate i lies in $[u - (c' \ln n)/n, u)$, where we relax notation and use u to also denote the ID of node u. We reiterate a well-known argument: since IDs are u.a.r., the probability that two nodes are separated by more than a $c('\ln n)/n$ distance is at most $(1 - (c' \ln n)/n)^n \leq 1/n^{c'}$. Therefore, if $h_1(w, i)$ lies outside of this interval, then w.h.p. u is not the successor of $h_1(w, i)$ and cannot be a valid neighbor. In this case, the request is safely rejected.

If $h_1(w, i)$ does lie within the interval, then u performs a search on $h_1(w, i)$. If this returns u, then u accepts the request. Given the guarantee on almost-everywhere routing in old quorum graphs, with probability at least $1 - O(1/\log^{k-c} n)$, this acceptance is correct (recall Lemma 4). In other words, the probability of accepting an erroneous neighbor request is at most $1/\text{poly}(\log n)$ where we can tune this polynomial.

How many neighbor requests with a valid $h_1(w, i)$ does w receive? Given that $h_1(w, i)$ is u.a.r. (given the random oracle assumption) it is valid with probability $(c' \ln n)/n$. By Property 3, $i = O(\text{poly}(\log n))$ and so

over at most n nodes, there are $O(\log^2 n)$ valid requests received by w; this is tight by a standard Chernoff bound.

It follows that w accepts at most $O(\log^2 n)/\text{poly}(\log n) = o(\log \log n)$ erroneous requests in expectation so long as our constant k is sufficiently large.

State Cost of w's Neighbors. This is straightforward: recall by property P3 of the input graph G, that w links to $O(|L_w|)$ other nodes. Therefore, in each quorum graph, w links to $O(|L_w|) = O(\text{poly}(\log n))$ quorums as neighbors.

State Cost of Neighbor Requests. Via Lemma 5, properties P1 and P3 guarantees that w can determine independently whether it should indeed be a neighbor of some node u, and there are at most $\text{poly}(\log n)$ such nodes u. Using the old quorum graphs, wherein almost-everywhere routing is guaranteed w.h.p., w initiates a search to check that it should indeed be a neighbor. With a tunable probability at least $1 - O(1/\log^{k-c} n)$ node w can detect if the request is erroneous. Therefore, in expectation, the number of erroneous acceptances is at most $o(|L_w|)$ so long as our constant k is sufficiently large (recall Lemma 4). □

B Generating Global Random Strings

Generating Global Random Strings. During epoch i, each node w forms a ***solution set*** of the $d_0 \ln n$ smallest random strings $\boldsymbol{R_i^w}$ for some constant $d_0 \geq 1$. Over epoch i, all good nodes generate random strings and the smallest are collected independently by each node w to create R_i^w. To generate a string in epoch i, a node w uses a string r_{i-1} — the globally-known string from the previous epoch — and an individually generated random string, $\boldsymbol{s_w}$, to compute the ***output*** $\boldsymbol{t_w} = h(s_w \oplus r_{i-1})$.

Bins and Counters. To facilitate our discussion of how to propagate strings and ease our subsequent analysis, we describe a system of bins and counters maintained by each good node w. The ***bins*** $\boldsymbol{B_j}$ correspond to intervals in the ID space where $B_j = [1/2^j, 1/2^{j-1})$ for $j = 1, 2, ..., b \ln(n\,T)$ where $b \geq 1$ is a sufficiently large constant. Since T is known and there are standard techniques for obtaining a constant-factor approximation to $\ln n$, calculating $\ln(nT) = \ln(n) + \ln(T)$ to within a constant factor is possible.[14]

Each bin B_j has an associated ***counter*** $\boldsymbol{C_j}$. Consider that w receives a string s_u with corresponding output t_u that falls within the interval defined by B_j; we say that B_j ***contains*** s_u. If t_u is smaller than the other values w has seen so far contained in B_j, and $C_j \leq c_0 \ln n$ for some sufficiently large constant $c_0 \geq 1$, then w increments C_j and forwards s_u onto its neighbors. After $C_j = c_0 \ln n$, no value landing within B_j is ever forwarded.

The intuition is that, if $c \ln n$ strings are found with "record-breaking" outputs in B_j, then w.h.p. smaller strings exist with outputs belonging to B_{j+1}. In other words, those strings corresponding to B_j will not be candidates a globally-known string and so they can be ignored.

Protocol for Propagating Strings. The propagation of strings is broken into ***phases*** which make up the first half of an epoch. We describe the protocol for good nodes (although bad nodes can deviate arbitrarily).

Phase 1 executes over steps 1 to $T/2 - 2d' \ln n$ for a constant $d' > 1$ of the current epoch i. Over this time, each node w generates random strings with associated outputs. After Phase 1 ends, nodes no longer generate new random strings.

Phase 2 begins at step $T/2 - 2d' \ln n + 1$ and runs for $d' \ln n$ steps. Each node w (using its quorum Q_w) selects the string $s_w^{\min}$ with the smallest output $t_w^{\min}$ that was generated in Phase 1, and then sends $s_w^{\min}$ its neighbors. Node w updates the corresponding bin and counter, as described earlier.

Each neighbor u verifies $s_w^{\min}$. Using $t_w^{\min}$, node u decides whether to forward $s_w^{\min}$ to its own neighbors (except for w) and, if so, updates the corresponding bin and counter; otherwise, u ignores this value. At the

[14] A standard technique for estimating $\ln n$ to within a constant factor is as follows. For nodes with u.a.r. IDs, the distance $d(u, v)$ between any two nodes u and v satisfies $\frac{1}{c'' n^2} \leq d(u, v) \leq \frac{c' \ln n}{n}$ w.h.p. that can be tuned depending only on sufficiently large positive constants c', c'' Therefore, with high probability, $\ln(\frac{1}{d(u,v)}) = \Theta(\ln n)$ and this holds even when an adversary decides to omit its nodes in the ID space (see Chapter 4 in [48]).

end of Phase 2, each node w selects the string with the smallest output it has seen so far; this is denoted by $\boldsymbol{s_w^{i*}}$.

Phase 3 starts at step $T/2 - d' \ln n + 1$ and runs for the final $d' \ln n$ steps. Over these steps, nodes no longer generate new strings, although they will still propagate them according to the above rules.

At the end of the phase, each node w creates its solution set R_i^w in the following way. Node w finds the largest j for which B_j contains at least one element. Then, w takes the union of subsequent bins for decreasing j until there are $d_0 \ln n$ elements; the collection of these elements form R_i^w.

This concludes the propagation protocol. We note that immediately (starting at step $T/2+1$) node w will start generating a new ID signed with the string s_w^{i*} chosen in Phase 2.

Discussion. The adversary may prevent good nodes from agreeing on the same solution set. As mentioned in Subsection 4.1, a $1/\text{poly}(\log n)$-fraction may be unable to partake in the propagation process even with our robust routing, and their loss is already incorporated into our analysis in Subsection 3.2. Therefore, we address the ***giant component*** of $(1 - 1/\text{poly}(\log n))n$ good nodes that can reach each other (and this set of nodes is implied by the terminology "good nodes").

The critical source of disagreement between good nodes is that the adversary may delay releasing a string s' (or multiple strings) with a small output. For example, if this occurs right before the end of Phase 2, then only a subset of good nodes receive s' and their respective solution sets differ from the other good nodes. In this way, two good nodes u and w can choose different strings s_u^{i*} and s_w^{i*}.

We sketch how this disagreement is handled, but first we address the simpler case where there is no adversarial interference.

With No Adversary: Note that the propagation of a string in the giant component requires at most $d' \ln n$ steps. Therefore, since all nodes send their string at the beginning of Phase 1, then by the end of Phase 2, all nodes accept the same set of strings and agree on the minimum string.

Furthermore, in Phase 3, nothing will occur (since no strings are released late) and so any nodes w and u are guaranteed w.h.p. to have $R_i^w = R_i^u$. What are the outputs corresponding to these solution sets? There are $\Theta(n)$ nodes computing for $\Theta(T)$ steps, so the smallest output in a set R_i^w is $\Theta(\frac{1}{nT})$ and w.h.p. no larger than $O(\frac{\ln n}{nT})$.

With an Adversary: The adversary can propagate a string s' with a small output late in Phase 2.[15] If w receives s' while u does not, then $R_i^w \neq R_i^u$. We argue that (1) the size of each solution set remains bounded by $\Theta(\ln n)$, and (2) that the string s_w^{i*} used by each good node w belongs to every other good nodes' solution set; these two properties enable efficient and correct verification (described below).

How many solutions s' could w receive and add to R_i^w? As noted above, this solution set will hold outputs of value $O(\frac{\ln n}{nT})$. Since the adversary has bounded computational power of βn, w.h.p. there cannot be more than $d'' \ln n$ solutions with output value $\Theta(\frac{1}{nT})$ for some constant $d'' > 0$. This is true even if the adversary computes over the entire epoch. We set the constant c_0 used in the bin counters such that $c_0 \geq d''$ in order to make sure that no smallest values are omitted.

Now assume that w selects s_w^{i*}, but that s_w^{i*} is not present in good node u's solution set R_i^u; we will derive a contradiction. If s_w^{i*} originated from a good node, then u received s_w^{i*} by the end of Phase 3 since $2d' \ln n$ steps is more then sufficient for the propagation of a string in the giant component. Else, s_w^{i*} originated from the adversary. Since s_w^{i*} was held by w by the end of Phase 2, the addition $d' \ln n$ steps in Phase 3 would have allowed s_w^{i*} to reach u and be added to R_i^u. In either case, this yields the contradiction.

Finally, what is the message complexity of the propagation protocol? Recall that for each bin, the associated counter restricts to $O(\ln n)$ the number of times a node forwards a string to its neighbors. Given that there are $O(\ln(nT))$ bins, the total number of times a node can forward a string is $O(\ln(n)\ \ln(nT))$. The number of messages sent between any pair of neighboring quorums is $O(|Q|^2) = O((\log\log n)^2)$ and the degree in

[15] The adversary may also delay a string from a good node outside the giant component, which amounts to the same problem since the adversary controls when this string is released into the giant component.

the quorum graph is $O(\text{poly}(\log n))$. Therefore, the total message complexity over $O(n)$ nodes is $\tilde{O}(n \ln T)$ where $\tilde{O}$ accounts for $\text{poly}(\log n)$ terms.

The above discussion supports the following:

Lemma 12. *With high probability, the protocol for propagating strings (i) guarantees that, for each good node w in the component, s_w^{i*} is contained within the solution set of every good node in the component, (ii) $|R_i^w| = O(\ln n)$, and (iii) has message complexity $\tilde{O}(n \ln T)$.*

Verifying IDs. For simplicity, our discussion of ID verification in Subsection 4.1 assumed that a single r_{i-1} was agreed upon. However, not much changes when using solution sets.

To generate an ID for use in epoch $i+1$, node w uses s_w^{i*} to sign its ID. By the above discussion, we are guaranteed w.h.p. that s_w^{i*} belongs to the solution set of each good node. Therefore, a good node u that wishes to verify w's new ID checks whether this ID was signed by *any* of the strings in R_i^u; this requires checking only $O(\ln n)$ elements by the above discussion.

C Membership in L_u

In our discussion of the properties of the input graph in Section 1.3, we made the following statement in $P3$: *"... there are $O(\text{poly}(\log N))$ nodes u whose IDs dictate $w \in L_u$"*. We discuss this here further using Chord as an example; however, the same property holds for the other input graphs we specify in Section 1.3.

We consider the version of Chord where IDs are in $[0, 1)$. Node u has neighbors (in its "Finger Table") that are found by taking points 2^{-i} and linking to the successor of each such point, for $i = 1, ..., O(\lg(N))$. Therefore, if u is claiming w as a neighbor, then w can examine the index i and immediately determine if $u + 2^{-i}$ is "close enough" to w. For example, if $u + 2^{-i} = 0.5$, but $w = 0.9$, then clearly the successor $u + 2^{-i}$ will not be w and the request is erroneous.

How close is "close enough" such that w must perform a search? Since IDs are u.a.r. in $[0, 1)$, it is easy to see that w.h.p. the largest interval between any two nodes is $\Theta(\log N/N)$. Therefore, if $u + 2^{-i}$ is outside of the subinterval $[w - \Theta(\log N/N), w)$, then w can ignore the request. Otherwise, it must perform a search to see if the successor of $u + 2^{-i}$ is indeed w.

How many nodes have a neighbor link that falls into this interval $[w - \Theta(\log N/N), w)$ and cannot be rejected out of hand? Again, a standard argument can be made that $O(\log N)$ nodes fall within any interval of this size, and this is tight w.h.p. by a Chernoff bound; a request from each such node must be checked via a search. Since the degree of each node has degree $\text{poly}(\log N)$, this means that only $O(\text{poly}(\log N))$ nodes can make neighbor requests as claimed.

D Bootstrapping Quorums

A standard assumption in the literature is that a node knows how to contact another node already in the system in order to be bootstrapped. In the absence of an adversary, this seems plausible and the assumption holds true in practice.

The bootstrapping issue is less clear in a Byzantine setting and we consider it an open challenge, although not within the scope of our work here. We can imagine that a node might know (i.e. have IP addresses and port numbers) for an entire quorum which can then act as Q_{boot}. However, it is unclear how this information would be provided.

If the information is advertised on a server, then this becomes a point of attack. Alternatively, if this information is hard-coded into the application that is downloaded onto the node, then we must rely on someone to do the hard-coding. Or perhaps another distributed system is in place to facilitate bootstrapping akin to Vuze for BitTorrent, but then that system must also be Byzantine fault tolerant.

Note that these issues arise whether all quorums are good, or "almost all" quorums are good. We conjecture that a solution is possible via PoW — having a quorum be able to offer a bootstrapping service only if it solves a puzzle — but we leave this issue to future work.