Making Chord Robust to Byzantine Attacks

Amos Fiat * Jared Saia † Maxwell Young †

That which is not good for the swarm is not good for the bee either.

Marcus Aurelius, *Meditations*, book VI verse 54

Abstract

Chord is a distributed hash table (DHT) that requires only $O(\log n)$ links per node and performs searches with latency and message cost $O(\log n)$, where n is the number of peers in the network. Chord assumes all nodes behave according to protocol. We give a variant of Chord which, for any fixed $\epsilon_0 > 0$, is resilient to $(1/4 - \epsilon_0)z$ Byzantine nodes joining the network over a time period during which 1) there are always at least z total nodes in the network and 2) the number of correct peers joining and leaving is no more than z^k for some tunable parameter k. We assume there is an omniscient and computationally unbounded adversary controlling the Byzantine peers and that the IP-addresses of all the Byzantine peers and the locations where they join the network are carefully selected by this adversary.

Our notion of resilience is rather strong in that we not only guarantee that searches can be performed but also that we can enforce any set of "proper behavior" such as contributing new material, etc. In comparison to Chord, the resources required by this new variant are only a polylogarithmic factor greater in communication, messaging, and linking costs.

1 Introduction

A distributed hash table (DHT) is a structured peer-to-peer network which provides for scalable and distributed storage and lookup of data items (see e.g. [22, 19, 25]). Because peer-to-peer networks have little to no admission control, there has been significant effort in designing DHT's which are robust to *Byzantine* faults. When a peer suffers a Byzantine fault it is assumed to be controlled by an omniscient adversary who uses that peer to try to disrupt the network. The standard attack model considered is as follows. There is an instantaneous attack during which each peer in the network suffers a Byzantine fault independently at random with constant probability (less than 1/2). We will refer to this type of attack as a *random Byzantine attack*. Several DHT's have been designed which provide robust storage and lookup of data items, even in the face of a random Byzantine attack [7, 17, 10].

While this past work is encouraging, the random Byzantine attack model is unsatisfying. In particular, it seems that a much more likely attack scenario is that an adversary

*Department of Computer Science, Tel Aviv University; email: fiat@math.tau.ac.il

†Department of Computer Science, University of New Mexico, Albuquerque, NM 87131-1386; email: saia@cs.unm.edu, young@cs.unm.edu. This research was partially supported by NSF grant CCR-0313160 and Sandia University Research Program grant No. 191445.

will cause a stream of Byzantine peers to join the network and carefully choose the IP-addresses of these Byzantine peers[1] and where they join the network in order to place them at critical locations in the network. To better address this scenario, we introduce a new attack model, which we call the *Byzantine join attack*. Under this attack, $(1/4-\epsilon_0)z$ Byzantine nodes join the network over a time period during which 1) there are always at least z total nodes in the network and 2) the number of correct peers joining and leaving is no more than z^k for some tunable parameter k. We assume there is a computationally unbounded adversary controlling the Byzantine peers and that the IP-addresses of all the Byzantine peers are selected by this adversary. We further assume that the adversary possesses full knowledge of the network topology, protocols, where data is stored, etc., and that the peers controlled by the adversary can actively collude to disrupt the network.

1.1 Our Contributions

In this paper, we describe a variant of Chord, *S-Chord*, which is robust to the Byzantine join attack. Define a *z-good* interval to be a time interval during which: 1) the number of total peers in the network is always at least z; 2) the number of Byzantine peers joining the network is no more than $(1/4 - \epsilon_0)z$ for some $\epsilon_0 > 0$; and 3) the number of correct peers joining and leaving during this time interval is no more than z^k for some tunable parameter k. Theorem 1 states the main result of this paper.

Theorem 1. *During any z-good interval, the following properties hold for S-Chord with high probability (specifically with probability of error polynomially small in z)*

- *All functionality of Chord is preserved.*
- *We can enforce a rule-set for all peers in the network.*
- *For n peers in the network, the resource costs are as follows:*
  - $O(\log n)$ *latency and expected* $O(\log^2 n)$ *messages sent per lookup operation.*
  - $\Theta(\log n)$ *latency and* $\Theta(\log^3 n)$ *messages sent per peer join operation.*
  - $O(\log^2 n)$ *links stored at each peer.*

In addition to being robust to the Byzantine join attack, S-Chord is also robust to the random Byzantine attack. Aside from robustness to this new, more realistic attack model, the other new contributions of S-Chord are as follows.

- S-Chord can enforce a set of rules describing "proper behavior" such as: "For every 20 search that a peer issues, that peer must service one search request". In particular, the consequences of not obeying the rules will be disconnection from the network. To the best of our knowledge, S-Chord is the first p2p network with this property.
- S-Chord is based on Chord and thus inherits many of Chord's good properties. Moreover, we feel that the general techniques used in this paper can be applied to a wide-range of other DHT's.

[1] i.e. by spoofing

- S-Chord requires $\Theta(\log^2)$ messages for lookups in expectation. Previous DHT's which are robust to the random Byzantine attack require $\Theta(\log^3 n)$ messages. Under the additional assumption of a computationally bounded adversary, we also show how we can ensure only an expected constant factor increase in the number of bits sent when sending large messages through the network.

1.2 Related Work

Recent years have witnessed the advent of large scale real-world peer-to-peer applications such as Gnutella, Napster, Kazaa, Morpheus, BitTorrent, and many others. Several distributed hash tables (DHTs) have been introduced which are provably robust to random peer deletions (i.e. fail-stop faults) [19, 25, 20, 22, 12, 1, 9].

We are aware of only three results which deal with the more challenging problem of designing DHTs which are robust to Byzantine faults. All three of these results are robust only to the random Byzantine attack described earlier. Fiat and Saia describe a DHT which uses expander graphs and a butterfly network to achieve robustness to this attack [7]. Unfortunately, this DHT is not fully dynamic in the sense that it does not easily allow for significant changes in network size. Naor and Wieder describe a much simpler DHT which is robust to the random Byzantine attack and is also fully dynamic [17]. Hildrum and Kubiatowicz describe how to modify two popular DHTs, Pastry [20] and Tapestry [25], in order to make them robust to the random Byzantine attack [10]. Their modified DHTs are fully dynamic.[2] In all three of these results, lookups have $\Theta(\log n)$ latency and require $\Theta(\log^3 n)$ messages. In this paper, we make use of ideas from all three of these results.

Our DHT makes use of secure multiparty computation in order to choose random IDs for joining peers by consensus. There is a significant body of work in the area of secure multiparty computation (see e.g. [23, 8, 3, 5, 2, 4, 21, 18, 11]). Section 4 and Appendix C describe in detail how we use these results.

The rest of this paper is organized as follows. Section 2 gives a high level overview of S-Chord. Section 3 describes with the protocol $SUCCESSOR$ which allows peers in S-Chord to publish and lookup content. Section 4 describes the protocol $JOIN$ which is used to allow new peers to join the network. Section 5 provides an algorithm that reduces the message complexity of $SUCCESSOR$ from $O(\log^3 n)$ worst case to $O(\log^2 n)$ in expectation. Section 6 gives an algorithm that allows $SUCCESSOR$ to incur only an expected constant factor increase in the number of bits sent over what is required for Chord. This result assumes a computationally bounded adversary. We conclude and suggest some avenues of future research in Section 7. The Appendix contains all proofs for these sections as well as other details omitted due to space constraints.

2 Overview

2.1 Chord

We now briefly describe Chord [22].[3] For convenience, we will assume that the "key space" of Chord is scaled so it is in the range $(0, 1]$ and will think of Chord as a circle

[2] We emphasize here that S-Chord is also fully-dynamic.

[3] For ease of exposition, our description will defer slightly from that of [22], but will not be fundamentally different.

with unit circumference, which we will call the *unit circle*. All of the peers in Chord have identifiers (or IDs for short) which are points on the unit circle that we call *peer points*. Chord provides one basic operation: *successor*(). For a point k on the unit circle, *successor*(k) returns the peer, p, whose peer point minimizes the clockwise distance between k and p. Typically, k represents a key for some data item and *successor*(k) is the peer responsible for storing that data item. Thus, the *successor*() operation provides for easy storage and lookups of data items.

We now briefly sketch how Chord implements the operation *successor*(). We assume that all peers in the network know some number m which is always greater than the number of peers in the network[4]. For a point p on the unit circle and integer i between 0 and $\log m - 1$, let $f(p, i)$ be the point $p + 2^i/m$. For each i between 1 and $\log m - 1$, each peer p maintains a link to the peer whose peer point is closest clockwise to the point $f(p, i)$. When a peer p links to a peer p', the peer p simply stores the IP address of p'. The number of unique peers that a peer p links to is $O(\log n)$. For points p and k on the unit circle, let $next(p, k)$ be the point in the set $\{f(p, 0), f(p, 1), f(p, 2), \ldots f(p, \log m - 1)\}$, which has closest clockwise distance to k.

We can now describe the *successor*() operation. Assume that some peer p calls *successor*(k) for some key k on the unit circle. If $next(p, k) = p$, then p already knows the successor of k: it is simply the closest clockwise peer to p. The search terminates by returning this peer. If $next(p, k) = p'$ where $p' \neq p$, then p forwards the search request to p'. The same procedure is repeated until the search terminates.

2.2 Notation

For any two points x and y on the unit circle, let $d(x, y)$ be the distance from x to y traveling clockwise along the perimeter of the unit circle (i.e. if $y \geq x$, then $d(x, y) = y - x$ else $d(x, y) = (1 - x) + y$). When referring to intervals or points on the unit circle, all addition is performed modulo 1. We will call a peer controlled by the adversary *faulty* and call a peer not controlled by the adversary (i.e. a peer that follows the protocol) *correct*.

2.3 S-Chord

In our protocol, peers do not get to choose their own ID's. Instead they are assigned, by our protocol, a random ID between 0 and 1 when they first join the network. Following convention, for a given peer p, we will frequently use p to refer both to the peer and to the ID of the peer. The precise meaning should be clear from context.

Central to our protocol is the notion of a *swarm*. For every point x on the unit circle, we define the swarm, $S(x)$, to be the set of peers whose ID's are located within a clockwise distance of $(C \ln n)/n$ of the point x on the unit circle (where C is a constant depending on our fault-tolerant parameters). For a given peer p, we will use $S(p)$ to mean the swarm associated with the peer p. All communication that p has with the DHT first passes through the swarm $S(p)$. Swarms, not peers, are the atomic functional units of our protocols. We say that a swarm is *good* if at least a 3/4 fraction of the peers in it are correct. Due to the fact that our protocol randomly assigns ID's to peers, we can guarantee with high probability that over a z-good time interval, all swarms will be good. Thus, we can say that even though many peers are not correct, all of the swarms

[4]In practice, m is the number of bits in the ID's of the nodes.

will be good. This fact is the basis for the robustness of our DHT.

Overview: We begin by assuming that all peers in the network know the values $\ln n$ and $(\ln n)/n$ exactly. Under this assumption, we present protocols for 1) obtaining content from network and sending messages (Section 3), 2) handling dynamic peer joins (Section 4) and 3) stabilization to handle peer deletions and the effects of changing network size on interval tracking (Section A.5). These protocols provide the same basic functionality of those used in Chord; however, they have been extended to work with swarms. In Appendix B, we give the required modifications to our protocols for the case where the peers do not know the values of $\ln n$ and $(\ln n)/n$.

2.4 Links Required

In this section, we state the links that each peer is required to maintain in our protocol. We will often make statements referring to some correct peer p maintaining links to all peers in an interval $[a, b]$ for $a, b \in (0, 1]$. Assume that this means p maintains links to all *correct* peers and those faulty peers of which p is aware. Every peer p maintains links to all peers in the following intervals.

- *Center Interval*: $Center(p)$ is the set of peers in the interval $[p - (2C \ln n)/n, p + (2C \ln n)/n]$.
- *Forward Intervals*: For all i between 1 and $\log m - 1$, $Forward(p, i)$ is the set of peers in the interval $[p + 2^i/m - (C \ln n)/n, p + 2^i/m + (C \ln n)/n]$.
- *Backward Intervals*: For all i between 1 and $\log m - 1$, $Backward(p, i)$ is the set of peers in the interval $[p - 2^i/m - (C \ln n)/n, p - 2^i/m + (C \ln n)/n]$.

Figure 1 illustrates links maintained in Chord and our enhanced DHT. A peer p keeps track of the links in the Center interval so that 1) p knows all peers in $S(p)$, 2) p knows all peers p' such that $p \in S(p')$ and 3) p is able to help compute the $SUCCESSOR$ algorithm described in Section 3. A peer p, keeps track of the Forward intervals so that is able to forward on requests for the $SUCCESSOR$ function. While in Chord, requests for a successor are forwarded to a single peer, in our system, they are forwarded to an entire swarm. A peer p, keeps track of the Backward intervals so that it is able to recognize legitimate requests sent during computations of the $SUCCESSOR$ function. In our protocol, we do not trust a peer to tell us its identifier (i.e. where it is located on the unit circle). Thus, a peer p specifically requires links to Backward intervals in order to keep track of the IDs of those peers who may legitimately send p messages. All messages sent to p from peers which are not in one of p's Backward intervals are ignored.

Lemma 1. *Let p and q be any two peers. Then, the following are true:*

- *If $p \in Center(q)$ then $q \in Center(p)$*
- *If $p \in Forward(q, i)$ for some i between 1 and $\log m - 1$, then $q \in Backward(p, i)$*
- *If $p \in Backward(q, i)$ for some i between 1 and $\log m - 1$, then $q \in Forward(p, i)$.*

Lemma 2. *With high probability, the number of peers any peer links to is $\Theta(\log^2 n)$.*

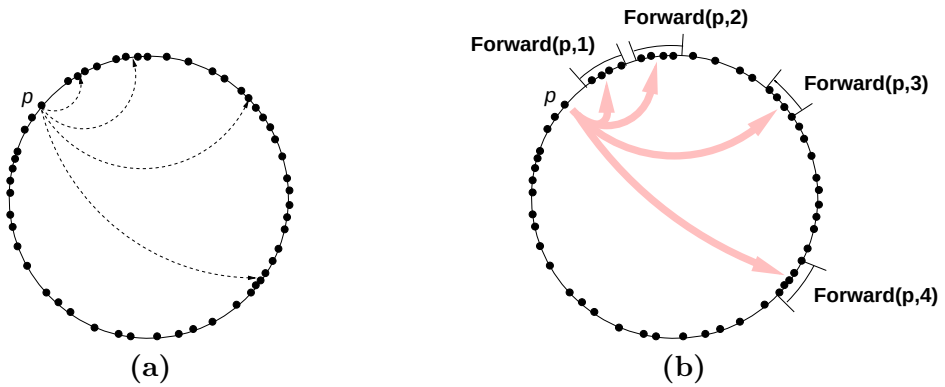

Figure 1: (a) Links maintained by peer p under Chord. (b) Links maintained by p to Forward intervals in our DHT. The grey arrows depict links from p to all peers in each interval shown. Peer p's links to its Center and Backward intervals are not shown here.

Algorithm 1 SUCCESSOR(p)

1: p sends a request for k to all peers in $S(p)$;
2: $S \leftarrow$ set of all peers in $S(p)$;
3: $x \leftarrow$ identifier of p;
4: **while** $(d(x,k) > (C \ln n)/n)$ **do**
5: $x' \leftarrow next(x,k)$;
6: All peers in S send the request for k to all peers in $S(x')$;
7: $S' \leftarrow$ set of all peers in $S(x')$ that received the above request from a majority of the peers in S;
8: $S \leftarrow S'$;
9: $x \leftarrow x'$;
10: **end while**
11: The peers in S send back pointers to all the peers in $S(k)$. These pointers are sent backwards along the same path, in the same manner, to the peer p;

3 Sucessor Protocol

Algorithm 1 gives the pseudocode for our robust $SUCCESSOR$ protocol which is analogous to the *successor* operation of Chord. Figure 2 illustrates a run of $SUCCESSOR$. For a point k on the unit circle, $SUCCESSOR(k)$ returns pointers to the peers in $S(k)$. As in Chord, k would typically represent a key for some data item. $SUCCESSOR(k)$ returns pointers to the *set* of peers responsible for storing that data item. Thus, the $SUCCESSOR$ operation provides for redundant storage and lookups of data items.

For a key k and peer p, $SUCCESSOR(k)$ works as follows when called by p. Peer p initially sends the request for k to all peers in $S(p)$. Let x equal the ID of p and S be $S(p)$. Until $d(x,k) \leq (C \ln n)/n$, the following loop repeats: the peers in S forward the request to all peers in $S(x')$ where $x' = next(x,k)$. Let S' be the set of peers in $S(x')$ which receive the request from a majority of peers in S. The loop now repeats with S set to S' and x set to x'. When the loop terminates, $d(x,k) \leq (C \ln n)/n$, so all peers in the set S have pointers to all peers in $S(k)$. These pointers to peers in $S(k)$ are then sent backwards along the same path, in the same manner, to the originating peer p.

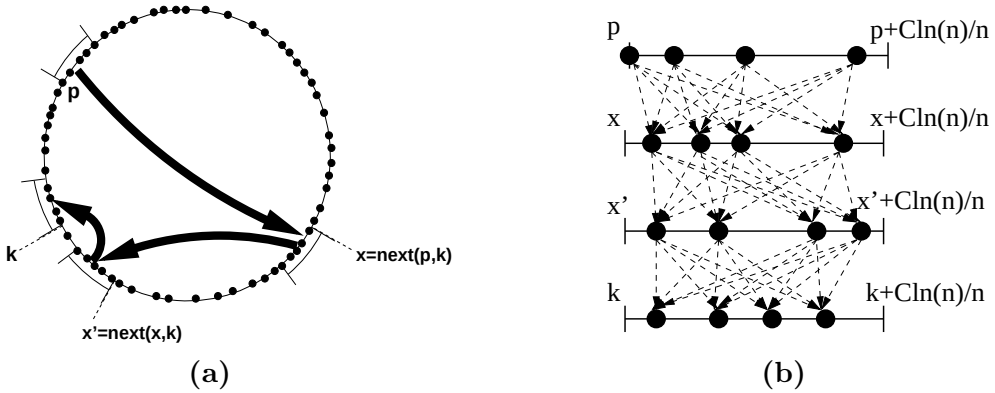

Figure 2: (a) An example of a call to $SUCCESSOR(k)$ by peer p. The dark arrows represent the path the search traverses.(b) A close-up the messages being sent.

The following lemma is true with high probability over any z-good time interval.

Lemma 3. *The following is true provided that all swarms in the DHT are good. For any key k and peer p, $SUCCESSOR(k)$ always returns pointers to all peers in $S(k)$ when called by peer p. Moreover, $SUCCESSOR$ has latency $O(\log n)$ and requires $O(\log^3 n)$ messages.*

For a given peer p, message m and an interval I on the unit circle, we define $SEND\_MESSAGE(m, I)$ to be an algorithm which allows p to send message m to all peers in the interval I. If I is of length $\Theta((\ln n)/n)$, it's straightforward to see how $O(1)$ calls to a modified $SUCCESSOR$ algorithm will create a $SEND\_MESSAGE$ algorithm with latency $O(\log n)$ and message cost $O(\log^3 n)$ (the detailed pseudocode is omitted). When writing the $JOIN$ protocol, we will make use of the $SEND\_MESSAGE$ algorithm.

We now describe conditions under which we can show that all swarms are good.

Lemma 4. *Assume that 1) all peer points are distributed uniformly at random on the unit circle; and 2) the fraction of faulty peers is no more than $1/4 - \epsilon$. Let k be any fixed integer and C be sufficiently large but depending only on k, then with probability at least $1 - 1/n^k$, the following statement is true. For any point x on the unit circle, the swarm $S(x)$ is good.*

We now provide a description of how S-Chord allows for the enforcement of a rule set on all peers in the system, provided that all swarms are good. The desired rule set must be known in advance by all correct peers. The rule set can be enforced by having the correct peers in a swarm act in concert to stop any prohibited behavior. For instance, if a faulty peer p attempts to abuse bandwidth resources by making excessive calls to $SUCCESSOR$, the correct peers in $S(p)$ can simply refuse to participate in the $SUCCESSOR$ calls after a certain pre-defined cut-off point.

4 Peer Joins

Pseudocode for the $JOIN$ algorithm is given in Algorithm 2 and an example run of the algorithm is illustrated in Figure 3. The $JOIN$ algorithm makes use of an algorithm

Algorithm 2 JOIN(p)

1: Peer p contacts some correct peer q which notifies $S(q)$ of p's request to join;
2: All peers in $S(q)$ come to consensus on a random number in $(0, 1]$ to be used as the ID for p, using the algorithm discussed in Appendix C;
3: All peers in $S(q)$ notify peers in $Center(p)$, using the $SEND\_MESSAGE$ algorithm, that p has joined the network;
4: All peers in $S(q)$ get pointers to the peers in $Center(p)$, using $O(1)$ calls to the $SUCCESSOR$ algorithm. All peers in $S(q)$ send these pointers to p;
5: The peers in $Center(p)$ send data items for all keys k such that $p \in S(k)$ and p then stores copies of these data items;
6: **for** $(i = (\log m - 3\log n)$ to $\log m)$ **do**
7: All peers in $S(p)$ use all-to-all communication with the set P of peers in $[p + 2^i/m, p + 2^i/m + (C \ln n)/n]$ telling them of p's arrival. All peers in S_1 then use all-to-all communication with the set of peers S_2 in $[p+2^i/m-(C \ln n)/n, p+2^i/m]$ telling them of p's arrival. In this way, all peers in $Forward(p, i)$ know about p;
8: In an almost identical fashion to Step 7, all peers in $S(p)$ get pointers to the peers in $Forward(p, i)$. All peers in $S(p)$ then send these pointers to p;
9: All peers in $S(p)$ use all-to-all communication with the set P of peers in $[p - 2^i/m, p - 2^i/m + (C \ln n)/n]$ telling them of p's arrival. All peers in S_3 then use all-to-all communication with the set of peers S_4 in $[p-2^i/m-(C \ln n)/n, p-2^i/m]$ telling them of p's arrival. In this way, all peers in $Backward(p, i)$ know about p;
10: In an almost identical fashion to Step 9, all peers in $S(p)$ get pointers to the peers in $Backward(p, i)$. All peers in $S(p)$ then send these pointers to p;
11: **end for**

which allows a good swarm to choose a random number in the range (0,1]. The $JOIN$ algorithm assumes that peer p knows some correct peer q. In the algorithm, p first contacts peer q with p's request to join the network. Peer q alerts $S(q)$ to this request and the peers in $S(q)$ first choose a random ID for p using the algorithm discussed in Section C. The peers in $S(q)$ then introduce p to the peers of $Center(p)$ ($Center(p)$ includes the peers in $S(p)$). All peers in $S(p)$ then find all the peers in p's Forward and Backward intervals. In addition, the peers in $S(p)$ introduce p to all peers, p', in the network such that p is now in a Center, Forward or Backward interval for p'.

Lemma 5. *The $JOIN$ protocol has the following properties with high probability:*

- *$JOIN$ has $\Theta(\log n)$ latency and $\Theta(\log^3 n)$ message complexity.*
- *After $JOIN$ completes, peer p knows all peers in its Center, Forward and Backward intervals.*
- *Let q be any peer with the property that p is in a Center, Forward or Backward interval for q. Then after $JOIN$ completes, q knows about the peer p.*
- *Assume, before p joins the network, that the fraction of faulty peers is no more than $1/4 - \epsilon$ and that all peer points are distributed uniformly at random on the unit circle. Then* after *p joins the network, all peer points are distributed uniformly at random on the unit circle.*

In Section A.5, we present the $STABILIZE$ protocol that allows peers to maintain current views of the network. As join operations occur, periodically running $STABILIZE$

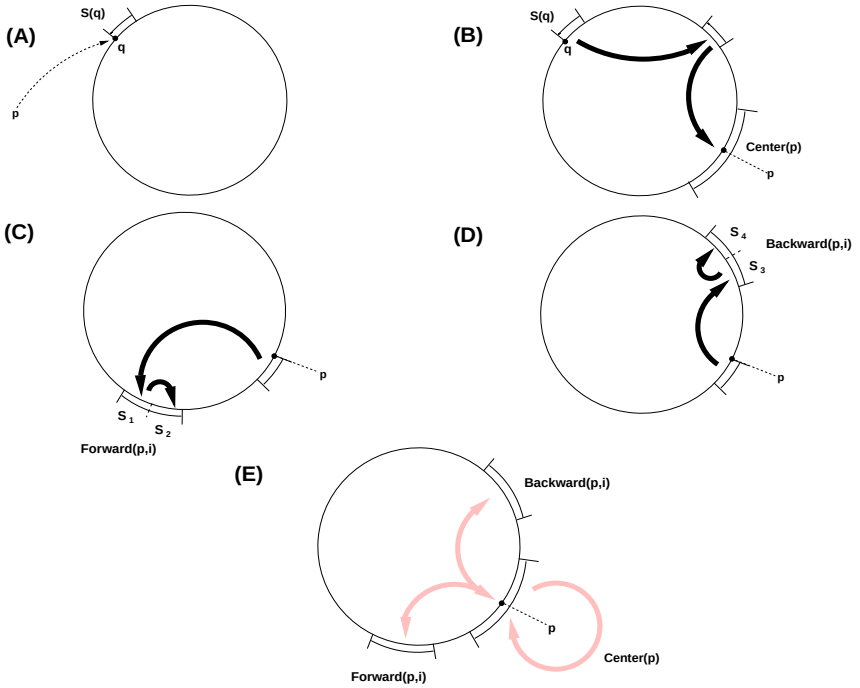

Figure 3: (A) Peer p contacts q asking to join the network. The peers in $S(q)$ generate a random number in (0,1] to be used as an identifier for p. (B) All peers in $S(q)$ notify all peers in $Center(p)$ that p is joining and send to p the identifiers of and pointers to all peers in $Center(p)$. (C) Peers in $S(p)$ obtain the identifiers of and pointers to the peers in the i^{th} Forward interval of p. All peers in this Forward interval are informed of p's arrival. This process is repeated with all Forward intervals of p. (D) Peers in $S(p)$ obtain the identifiers of and pointers to the peers in the i^{th} Backward interval of p. All peers in this Backward interval are informed of p's arrival. Again, this process is repeated with all Backward intervals of p. (E) Links established after the join protocol. The light colored arrows illustrate links between p and the peers in its Forward, Backward, and Center intervals. There are links between p and the peers in all of its Forward and Backward intervals although this is not shown in this figure.

keeps a peer p's pointers to its Forward, Backward, and Center intervals up to date. In the case of concurrent joins, temporary slowdowns in $SUCCESSOR$ operations may occur since pointers may be stale or absent. However, the periodic invocation of $STABILIZE$ will correct out of date pointers and allow for the $SUCCESSOR$ protocol to function normally.

5 $\Theta(\log^2 n)$ Expected Messages For SUCCESSOR

In this section, we show how to improve $SUCCESSOR$ so that it sends only $\Theta(\log^2 n)$ messages in expectation. We assume that all peers have a hash function h_1 which maps peer identifiers to the positive integers. We make the random oracle assumption about h_1 i.e. for any input, all outputs are equally likely. We assume that the number of peers in any swarm is $\Theta(\log n)$ and at least $C \log n$ for some fixed constant C. We also assume that all swarms have at least a 3/4 fraction of correct peers.

Algorithm 3 Message Sending Protocol

1: Each peer $x \in S_{j-1}$ sends a message to peer $y \in S_j$ iff

$$h_1(x) = h_1(y) \mod \log n$$

2: Each peer $y \in S_j$ accepts a message from peer $x \in S_{j-1}$ iff

$$h_1(x) = h_1(y) \mod \log n$$

3: Each peer $y \in S_j$, upon receiving messages from at least 2/3-rds of the peers that it would accept from, does majority filtering on all the messages received to decide which message if any to propagate to the next swarm.

Our algorithm for reducing message cost when sending from swarm S_{j-1} to swarm S_j is given in Algorithm 3. It assumes that swarm S_{j-1} wants to send a message to a swarm S_j (For ease of exposition, for a real number r, we will write r instead of $\lceil r \rceil$. It should be clear from context which is meant.). This algorithm is used in steps 6 and 7 of the SUCCESSOR pseudocode given in Algorithm 1. The proof of the following Lemma is given in Appendix A.6.

Lemma 6. *For C sufficiently large but depending only on k', the following is true with probability at least $1 - 1/n^{k'}$:*

- *All calls to SUCCESSOR succeed.*
- *All calls to SUCCESSOR send $\Theta(\log^2 n)$ messages in expectation.*

6 Expected Constant Factor Increase in Number of Bits

In this section we assume that peer p is trying to transmit a message m along the path $p, S_1, S_2, S_3, \ldots, S_l$ where S_i, $i = 1, \ldots, l$ are swarms. We assume that the adversary is polynomially bounded and that all swarms have at least a 3/4 fraction of correct peers.

Let $|m|$ be the number of bits in the message m. Peer p first encodes m into $\log n$ pieces $e_0, e_1, \ldots, e_{\ln - 1}$ with the following properties: 1) each piece has $O(|m|/\log n)$ bits and 2) m can be reconstructed from any 1/16-th fraction of the pieces. Any standard erasure code, such as tornado codes [14], can be used to create pieces with these two properties.

Peer p next creates fingerprints $f_1, f_2, \ldots, f_{\log n - 1}$ of all these pieces using a 1-way hash function, h_2 known by all the peers. For all $i = 1, \ldots, \log n - 1$, $f_i = h_2(e_i)$. Each of the fingerprints has $\log^2 n$ bits. This ensures that the probability that a random string maps to a fixed fingerprint is $1/n^{\log n}$. Thus it will take the adversary superpolynomial time to find a string which maps to one of the fingerprints.

Peer p sends all of the fingerprints to all of the peers in S_1. Then for all $j = 2, \ldots, l$, all peers in S_{j-1} send all of the fingerprints to all the peers in S_j and peers in S_j accept a fingerprint if and only if it was received from a majority of peers in S_{j-1}. This guarantees that all peers in the swarms $S_1, S_2, \ldots, S_l$ know all of the fingerprints. We will thus assume, in the protocol, described in this section, that a peer accepts a string s as some piece e_j iff $h_2(s) = f_j$.

Algorithm 4 Sending from S_{j-1} to S_j

1: Each peer in S_{j-1} sends the fingerprints $f_0, f_1, \ldots, f_{\ln n-1}$ to each peer in S_j.
2: The peers accept only those fingerprints that they receive from a majority of the peers in S_{j-1}. In the remainder of the algorithm, a peer in S_j only accepts a string s as some piece e_j if $h_2(s) = f_j$. .
3: **while** TRUE **do**
4: Peers in S_{j-1} come to consensus on a random integer r in $\{0, 1, \ldots \log n - 1\}$ using the protocol describe in Section C.
5: Let $P = \{x \in S_j | h_1(x) = r\}$. All peers in S_{j-1} send all of the pieces they currently hold to all peers in the set P.
6: All peers in P reconstruct the message m from the pieces received. From the message m, they then recompute the pieces $e_0, e_1, \ldots, e_{\log n-1}$.
7: For each $i = 1, \ldots, \log n - 1$, all peers in P send piece e_i to all peers $x \in S_j$ such that $h_1(x) = i \mod \log n$.
8: The peers in S_j come to consensus about whether they want a resend as follows:

1. Each peer $x \in S_j$ does the following. If $h_1(x) = i \mod \log n$ and x received piece e_i, x writes all peers in S_j that it received its piece.

2. Every peer $x \in S_j$ does the following. If x received messages indicating that at least $(3/4)|S_j|$ peers received their pieces, it tentatively sets an individual "resend" bit to 0 otherwise it sets this bit to 1.

3. All peers in Y do Byzantine agreement to come to consensus on the "resend" bit values set in the previous step.

9: The peers in S_j send to the peers in S_{j-1} the results of this consensus i.e. either that they want a resend or that they do not want a resend.
10: If the peers in S_{j-1} receive responses from more than 1/4 of the peers in S_j that they do not want a resend, the algorithm terminates.
11: **end while**

The first step of our protocol is simple. Peer p sends piece e_i to peer $x \in S_1$ iff $h_1(x) = i \mod \log n$. The general protocol where S_{j-1} sends to S_j for all $j = 2, \ldots, l$ is given as Algorithm 4 and illustrated in Figure 4. This protocol makes use of a Byzantine agreement protocol and the protocol for coming to consensus on a random number discussed in Section C. The proof of the following Lemma is given in Appendix A.7.

Lemma 7. *For any fixed k, for C sufficiently large but depending only on k, the following is true with probability $1 - 1/n^k$, if we run Algorithm 4 with $SEND\_MESSAGE$:*

- *All calls to $SEND\_MESSAGE$ succeed.*
- *All calls to $SEND\_MESSAGE$ have latency $O(\log n)$ in expectation.*
- *All calls to $SEND\_MESSAGE$ require $O(\log^4 n)$ messages to be sent in expectation.*
- *All calls to $SEND\_MESSAGE$ require $O(|m| \log n + \log^5 n)$ bits to be sent in expectation.*

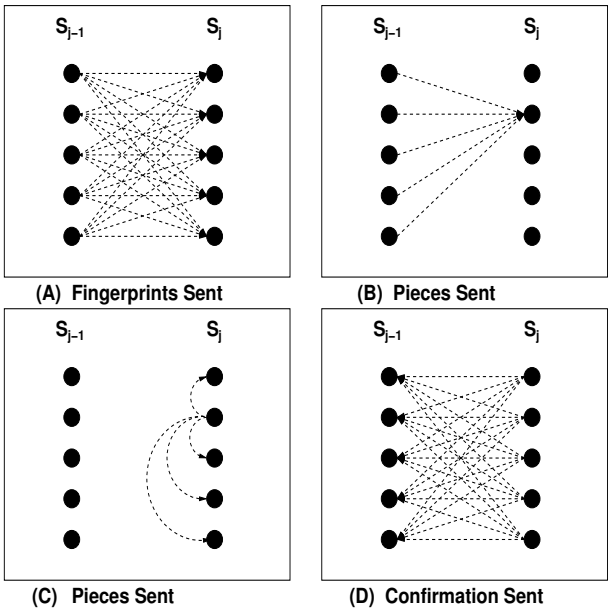

Figure 4: (A) All peers in S_{j-1} send fingerprints to all peers in S_j. Peers in S_j majority filter on these fingerprints. (B) All peers in S_{j-1} send all pieces of the message to those peers in S_j belonging to the set P described in Algorithm 4 (in this figure, P consists of only one peer). (C) The peer in P reconstructs the message, recomputes the pieces, and sends these pieces to the peers in S_j. (D) Peers in S_j come to a consensus and communicate with S_{j-1} as to whether or not they wish S_{j-1} to resend.

7 Conclusion

In this paper, we have introduced the Byzantine join attack, an attack model under which an omniscient adversary causes a large number of Byzantine peers to join a network. We assume that the adversary carefully chooses the IP-addresses of these peers and where they join the network in order to try to place them at critical locations. We have described, S-Chord, a variant of Chord that is provably robust to the Byzantine join attack. S-Chord also allows us to enforce a rule set on the peers in the network and thereby prevent undesirable behavior. In comparison to Chord's *successor*, this robustness is gained at the cost of an expected $\log n$ factor increase in the number of messages per $SUCCESSOR$ operation and a $\log n$ factor increase in the number of links stored per peer. We have also shown that if we make the additional assumption of a computationally bounded adversary and message sizes are large, $SUCCESSOR$ can be implemented with only an expected constant factor increase in the number of bits sent over what is required by Chord.

There are several problems that deserve future attention. First, can some of the techniques of this paper be adapted to be used in a practical peer-to-peer system? Second, it seems likely that $\Omega(\log^2 n)$ is a lower bound on the message complexity in the Byzantine model; however, can we prove this? Finally, can the methods we have presented also be applied to make other DHTs, such as CAN and Tapestry, resilient to Byzantine join attack?

References

[1] James Aspnes and Gauri Shah. Skip graphs. In *Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 384–393, 2003.

[2] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *Proceedings of the Twenty-Fifth ACM Symposium on the Theory of Computing (STOC)*, 1993.

[3] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computing. In *Proceedings of the Twentieth ACM Symposium on the Theory of Computing (STOC)*, pages 1–10, 1988.

[4] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. Asynchronous secure computations with optimal resilience. In *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 183–192, 1994.

[5] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC)*, pages 11–19, 1988.

[6] Ivan Damgård, editor. *Lectures on data security: modern cryptology in theory and practice*, volume 1561 of *Lecture Notes in Computer Science*. Springer, 1999.

[7] Amos Fiat and Jared Saia. Censorship resistant peer-to-peer content addressable networks. In *Proceedings of the Thirteenth ACM Symposium on Discrete Algorithms (SODA)*, 2002.

[8] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game - a completeness theorem for protocols with honest majority. In *Proceedings of the Nineteenth ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.

[9] Nicholar Harvey, Michael Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. Skipnet: A scalable overlay network with practical locality properties. In *Fourth USENIX Symposium on Internet Technologies and Systems(USITS)*, 2003.

[10] Kristen Hildrum and John Kubiatowicz. Asymptotically efiicient approaches to fault-tolerance in peer-to-peer networks. In *Proceedings of the 17th International Symposium on Distributed Computing*, 2004.

[11] Martin Hirt, Jesper Buus Nielsen, and Bartosz Przydatek. An asynchronous multiparty computation protocol. In Submission, 2004.

[12] M. Frans Kashoek and David R. Karger. Koorde: A simple degree-optimal distributed hash table. In *Proceedings of the Second International Workshop on Peer-to-Peer Systems (IPTPS)*, Berkeley, CA, 2003.

[13] Valerie King and Jared Saia. Choosing a random peer. In *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing (PODC)*, 2004.

[14] Michael G. Luby, Michael Mitzenmacher, M. Amin Shokrollahi, Daniel A. Spielman, and Volker Stemann. Practical loss-resilient codes. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 150 – 159, 1997.

[15] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. In *Proceedings of the Twenty-First ACM Symposium on Principles of Distributed Computing (PODC)*, 2002.

[16] Rajeev Motwani and Prbhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[17] 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)*, 2003.

[18] B. Prabhu, K. Srinathan, and C. Pandu Rangan. Asynchronous unconditionally secure computation: An efficiency improvement. In *INDOCRYPT 2002, Lecture Notes in Computer Science*, volume 2551, pages 93–107. Springer-Verlag, 2002.

[19] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2001.

[20] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329 – 350, 2001.

[21] K. Srinathan and C. Pandu Rangan. Efficient asynchronous secure multiparty distributed computation. In *INDOCRYPT 2000, Lecture Notes in Computer Science*, volume 1977, pages 117–129. Springer-Verlag, 2000.

[22] 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 2001 ACM SIGCOMM Conference*, 2001.

[23] Andrew Yao. Protocols for secure computations. In *Proceedings of the Twenty-Third IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 160–164, 1982.

[24] Kevin C. Zatloukal and Nicholas J. A. Harvey. Family trees: an ordered dictionary with optimal congestion, locality, degree, and search time. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete algorithms (SODA)*, 2004.

[25] 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.

A Appendix

A.1 Proof of Main Theorem

Theorem 1. *During any z-good interval, with high probability (specifically with failure probability polynomially small in z), the following properties hold:*

- *All functionality of Chord is preserved.*
- *We can enforce a rule-set for all peers in the network.*
- *For n peers in the network, the resource costs are as follows:*
  - $O(\log n)$ *latency and expected* $O(\log^2 n)$ *messages sent per lookup operation.*
  - $\Theta(\log n)$ *latency and* $\Theta(\log^3 n)$ *messages sent per peer join operation.*
  - $O(\log^2 n)$ *links stored at each peer.*

Proof. Consider any z-good interval and let $l = z^k$ be the number of insertion and deletion events during this interval. We first show that, with high probability, all swarms are good during the entire z-good interval. Let ξ_i be the event that some swarm is bad immediately after the i-th insertion or deletion event. Then we know that:

$$\begin{aligned} Pr(\cup_{i=1}^{l}\xi_i) &= Pr(\cup_{i=1}^{l}(\xi_i|\cap_{j=1}^{i-1}\bar{\xi_j})); \\ &\leq \sum_{i=1}^{l} Pr(\xi_i|\cap_{j=1}^{i-1}\bar{\xi_j}). \end{aligned}$$

We now seek to bound $Pr(\xi_i|\cap_{j=1}^{i-1}\bar{\xi_j})$ for any i. This is the probability that some swarm becomes bad after the i-th insertion or deletion event given that all swarms were good *before* this event. If all swarms were good before the i-th insertion or deletion event, then by Lemma 5 all peers that were inserted into the network from the start of the z-good interval until immediately after this i-th event, were inserted at random locations on the unit circle. Assuming that deletions of correct peers are not dependent on the locations of the correct peers on the unit circle, this implies that the correct peers are all distributed uniformly at random on the unit circle immediately after the i-th insertion or deletion event.

Now note that in the worst case, none of the Byzantine peers leave the network during the z-good interval. So we can assume that there are at most $(1/4-\epsilon_0)z$ Byzantine peers distributed uniformly at random on the unit circle and at least z total peers distributed uniformly at random on the unit circle. Thus, by Lemma 4, after the i-th insertion or deletion event, all swarms are good with probability $1 - 1/n^{k'}$ for any fixed constant k'. Since $n \geq z$, this implies that $Pr(\xi_i|\cap_{j=1}^{i-1}\bar{\xi_j}) \leq 1/z^k$. Plugging this into the above, we have

$$\begin{aligned} \sum_{i=1}^{l} Pr(\xi_i|\cap_{j=1}^{i-1}\bar{\xi_j}) &\leq \sum_{i=1}^{l} 1/z^{k'}; \\ &= z^k/z^{k'}; \\ &= z^{k-k'}. \end{aligned}$$

Thus for k' chosen sufficiently large but depending only on k, this probability can be made arbitrarily small. Provided that all swarms are good for the entire z-good interval, we can enforce a rule-set for all peers in the network. Moreover, Lemma 3 implies that all calls to successor will succeed and will have resource costs given in the theorem statement. The resource costs for the join operation follow immediately from Lemma 5. □

A.2 Proofs for Section 2: Linkage Structure

Lemma 1. *Let p and q be any two peers. Then, the following are true.*

- *If $p \in Center(q)$ then $q \in Center(p)$*
- *If $p \in Forward(q, i)$ for some i between 1 and $\log m - 1$, then $q \in Backward(p, i)$*
- *If $p \in Backward(q, i)$ for some i between 1 and $\log m - 1$, then $q \in Forward(p, i)$.*

Proof. If $p \in Center(q)$, then $p \in [q - (2C \ln n)/n, q + (2C \ln n)/n]$ which implies that $q \in Center(p)$. If $p \in Forward(q, i)$ for some i between 1 and $\log m - 1$, then $p \in [q + 2^i/m - (C \ln n)/n, q + 2^i/m + (C \ln n)/n]$. This implies that $q \in [p - 2^i/m - (C \ln n)/n, p - 2^i/m + (C \ln n)/n]$. Thus $q \in Backward(p, i)$. A similar argument shows that if $p \in Backward(q, i)$ for some i between 1 and $\log m - 1$, then $q \in Forward(p, i)$. □

Lemma 2. *With high probability, the number of peers any peer links to is $\Theta(\log^2 n)$.*

Proof. With high probability, when n peers are distributed uniformly at random on the unit circle, the shortest arc length between two peers is $\Theta(1/n^2)$ [13]. Thus with high probability, the number of actual forward and backward intervals each peer must keep track of is actually only $\Theta(\log n)$ (i.e. not $\Theta(\log m)$). By standard Chernoff bounds, we can show that with high probability, the total number of peers falling in any Center, Forward or Backward interval is $\Theta(\log n)$. The lemma then follows directly. □

A.3 Proofs for Section 3: $SUCCESSOR$ and Swarm Goodness

Lemma 3. *For any key k and peer p, $SUCCESSOR(k)$ always returns pointers to all peers in $S(k)$ when called by peer p. Moreover, $SUCCESSOR$ has latency $O(\log n)$ and requires $O(\log^3 n)$ messages.*

Proof. We first show that all peers in our network have the links required to follow the $SUCCESSOR$ protocol. First note that all peers in $S(x)$ must be able to send the request to all peers in $S(next(x, k))$. In other words, for any i between 1 and $\log m - 1$, all peers in the interval $[x, x + (C \ln n)/n]$ must know all peers in the interval $[x + 2^i/m, x + 2^i/m + (C \ln n)/n]$. This requirement is satisfied since each peer tracks Forward intervals and so knows, for all i between 1 and $\log m - 1$, all peers in the interval $[p + 2^i/m - (C \ln n)/n, p + 2^i/m + (C \ln n)/n]$. Note also that all peers in the interval $[next(x, k), next(x, k) + (C \ln n)/n]$ need to know all peers in the interval $[x, x + (C \ln n)/n]$ (so that they can majority filter). This requirement is satisfied since each peer tracks Backwards intervals and so knows, for all i between 1 and $\log m - 1$, all peers in the interval $[p - 2^i/m - (C \ln n)/n, p - 2^i/m + (C \ln n)/n]$. Finally, we note that the while loop of this protocol terminates when $d(x, k) \leq (C \ln n)/n$. At this point, all peers in $S(x)$ are expected to know all peers in $S(k)$. In other words, all peers in the interval

$[x, x + (C \ln n)/n]$ are expected to know all peers in the interval $[k, k + (C \ln n)/n]$ where $k \leq x + (C \ln n)/n$. Note also that all peers in the interval $[k, k + (C \ln n)/n]$ are expected to know all peers in the interval $[x, x + (C \ln n)/n]$. These requirements are satisfied since each peer tracks a Center interval and so knows all peers in the interval $[p - (2C \ln n)/n, p + (2C \ln n)/n]$.

We now show that $SUCCESSOR(k)$ always returns pointers to all peers in $S(k)$ when called by any peer p. Initially, p sends the request for k to $S(p)$. By Lemma 4, $S(p)$ is good with high probability. Thus, in the first iteration of the while loop, all peers in $S(x')$ will receive the request for k from a majority of peers in $S(p)$. A simple inductive argument then shows that in any iteration of the while loop, all peers in $S(x')$ will receive the request for k from a majority of the peers in S. This implies that when $d(x, k) \leq (C \ln n)/n$, a majority of the peers in $S(x)$ will be good and have the request for k. These peers will then retrieve pointers to all the peers in $S(k)$. The proof that these pointers arrive safely back to the peer p follows from a symmetric argument.

We now show that the latency is $O(\log n)$. Note that, by construction of the *next* function, each iteration of the while loop reduces the distance between x and k by at least a factor of two. The loop terminates when the distance between x and k is no more than $(C \ln n)/n$. In the worst case, this distance initially is 1. Thus, if we let i be the number of iterations, we require that $1/2^i \leq (C \ln n)/n$. This requires that $i \geq \log n - C \log \ln n$, so the number of iterations and thus the latency is always $O(\log n)$. In each iteration of the while loop, all peers in the set S forward the request for k to all peers in the set $S(x')$. Both S and $S(x')$ are of size $O(\log n)$, so the number of messages sent in each iteration is $O(\log^2 n)$. Since there are $O(\log n)$ iterations, the total number of messages sent is $O(\log^3 n)$ □

We now present the lemmas and proofs related to showing swarm-goodness.

Lemma 8. *For any positive integer k and positive ϵ, for C sufficiently large but depending only on k and ϵ, with probability at least $1 - 1/n^k$ the following statement is true. For any interval I of length $(C \ln n)/n$, I contains at least $(1 - \epsilon)C \ln n$ peers.*

Proof. Let ξ be the event that some interval of length I contains less than $(1 - \epsilon)C \ln n$ peers. If such an interval exists, there must also be an interval I' with the same properties and the additional property that I' is open on its counterclockwise end starting at some peer point p. Thus, to bound the probability of ξ, we need only consider the existence of such an interval.

Consider a fixed peer p and let I_p be an interval which is open on its counterclockwise end starting at some peer point p and which has length $(C \ln n)/n$. Let X_p be a random variable giving the number of peers in I_p. By linearity of expectation, $E(X_p) = ((n-1)/n)(C \ln n)$, which implies that $(1/2)C \ln n \leq E(X_p) \leq C \ln n$. Using Chernoff bounds, we can say that for all $\delta > 0$:

$$Pr[X_p < (1 - \delta)E(X_p)] < e^{-\frac{\delta^2 E(X_p)}{3}}.$$

Setting $\delta = 1 - (n/(n-1))(1 - \epsilon)$, we get:

$$\begin{aligned} Pr[X_p < (1 - \epsilon)C \ln n] &\leq e^{-\frac{\delta^2 E(X_p)}{3}}; \\ &\leq e^{-\frac{\epsilon^2 (C \ln n)}{6}}. \end{aligned}$$

Where the last line follows since $E(X_p) \geq (1/2)(C \ln n)$ and for n large, $\delta^2 \geq \epsilon^2$. Now if we do a union bound over all peers p, we get that:

$$\begin{aligned} Pr(\xi) &\leq ne^{-\frac{\epsilon^2(C \ln n)}{6}}; \\ &\leq e^{\ln n - \frac{\epsilon^2(C \ln n)}{6}}; \\ &\leq 1/n^k. \end{aligned}$$

Where the last line follows provided that C is sufficiently large. □

Lemma 9. *For any fixed integer k and fixed ϵ, such that $0 < \epsilon < 4\epsilon_0$, for C sufficiently large but depending only on k, ϵ and ϵ_0, with probability at least $1 - 1/n^k$ the following statement is true. For any interval I of length $(C \ln n)/n$, I contains no more than $1/4(1-\epsilon)C \ln n$ faulty peers.*

Proof. Let ξ be the event that there is some interval I with length $(C \ln n)/n$ such that the number of faulty peers falling in I is greater than $1/4(1-\epsilon)C \ln n$. If such an interval exists, there must also be an interval I' such that I' is of length $(C \ln n)/n$, the number of faulty peers falling in I is more than $1/4(1-\epsilon)C \ln n$ *and* the counterclockwise endpoint of I' is the peer point for some faulty peer p. Thus, to bound the probability of ξ, we need only consider those intervals whose counterclockwise endpoint is the peer point for some faulty peer.

Consider a fixed faulty peer p and the interval I_p which starts at p's peer point and has length $(C \ln n)/n$. Let X_p be a random variable giving the number of faulty peers falling in I_p. By linearity of expectation, we can say that $E(X_p) \leq 1 + (1/4 - \epsilon_0)(C \ln n)$ and $E(X_p) \geq (1/4 - \epsilon_0)(C \ln n)$. To use Chernoff bounds, we first want to choose a δ such that $Pr(X_p \geq 1/4(1-\epsilon)C \ln n) \leq Pr(X_p \geq (1+\delta)E(X_p))$. This requires that $(1+\delta)E(X_p) \leq 1/4(1-\epsilon)C \ln n$ which requires that:

$$\begin{aligned} \delta &\leq \frac{1/4(1-\epsilon)C \ln n}{E(X_p)} - 1; \\ &\leq \frac{1/4(1-\epsilon)C \ln n}{(1/4 - \epsilon_0)C \ln n} - 1; \\ &\leq \frac{1/4 - \epsilon/4}{1/4 - \epsilon_0} - 1. \end{aligned}$$

Set $\delta = \frac{1/4-\epsilon/4}{1/4-\epsilon_0} - 1$. Then since $\epsilon < 4\epsilon_0$, we know that $\delta > 0$. Hence using Chernoff bounds, we can say that:

$$\begin{aligned} Pr(X_p \geq 1/4(1-\epsilon)C \ln n) &\leq Pr(X_p \geq (1+\delta)E(X_p)); \\ &< e^{-\frac{\delta^2 E(X_p)}{3}}; \\ &\leq e^{-\frac{\delta^2(1/4-\epsilon_0)(C \ln n)}{3}}. \end{aligned}$$

Where the second line follows due to Chernoff bounds. Now if we do a union bound over all peers p, we get that :

$$\begin{aligned} Pr(\xi) &\leq ne^{-\frac{\delta^2(1/4-\epsilon_0)(C\ln n)}{3}}; \\ &\leq e^{\ln n - \frac{\delta^2(1/4-\epsilon_0)(C\ln n)}{3}}; \\ &\leq 1/n^k. \end{aligned}$$

Where the last line follows provided that C is sufficiently large but dependent only on k, ϵ and ϵ_0. □

Lemma 4. *For any fixed integer k, for C sufficiently large but depending only on k, with probability at least $1 - 1/n^k$, the following statement is true. For any point x on the unit circle, the swarm $S(x)$ is good.*

Proof. Let ϵ be fixed such that $0 < \epsilon < 4\epsilon_0$ and let k be any positive integer, then for C chosen sufficiently large but depending only on ϵ and k, with probability $1 - 1/n^k$, the following statement is true by Lemma 8 and Lemma 9. For any interval I of length $(C \ln n)/n$, the number of peers in I is at least $(1-\epsilon)C\ln n$ and the number of faulty peers in I is no more than $1/4(1-\epsilon)C\ln n$. But these two facts imply that there is no more than a $1/4$ fraction of faulty peers in I. Since a swarm is simply all the peers in an interval of length $(C\ln n)/n$, this implies that for any point x on the unit circle, the swarm $S(x)$ is good. □

A.4 Proof for Section 4: *JOIN*

Lemma 5. *The JOIN protocol has the following properties with high probability:*

- *JOIN has $\Theta(\log n)$ latency and $\Theta(\log^3 n)$ message complexity.*
- *After JOIN completes, peer p knows all peers in its Center, Forward and Backward intervals.*
- *Let q be any peer with the property that p is in a Center, Forward or Backward interval for q. Then after JOIN completes, q knows about the peer p.*
- *Assume, before p joins the network, that the fraction of faulty peers is no more than $1/4 - \epsilon$ and that all peer points are distributed uniformly at random on the unit circle. Then* after *p joins the network, all peer points are distributed uniformly at random on the unit circle.*

Proof. The peers of $S(q)$ generate an identifier for peer p using the protocol discussed in Appendix C. This incurs $\Theta(\log n)$ latency and requires $\Theta(\log^3 n)$ messages to be sent.

The *SUCCESSOR* protocol is then called $O(1)$ times by $S(q)$ to obtain pointers to all peers in $Center(p)$ and to inform all peers in $Center(p)$ of p's entry into the network; this accounts for the $O(\log n)$ latency. Each peer p' in $S(p)$ maintains links to all peers in $Forward(p,i) = [p' + 2^i/m - (C\ln n)/n, p' + 2^i/m + (C\ln n)/n]$. Therefore, all peers in $S(p)$ can both obtain pointers to the set S_1 of peers in $[p + 2^i/m, p + 2^i/m + (C\ln n)/n]$ as well as directly notify these peers of p's existence using all-to-all communication. Via its Backward interval links, each peer p'' in S_1 has pointers to the set S_2 of peers in $[p + 2^i/m - (C\ln n)/n, p + 2^i/m]$. In this way, the peers in S_1 can easily obtain the pointers to the peers in S_2 and notify them of p's existence using all-to-all communication.

Therefore, obtaining pointers for $Forward(p,i)$ and alerting the peers of $Forward(p,i)$ of p's existence requires $O(1)$ latency and $O(\log^2 n)$ messages. The same argument and complexity analysis applies to method by which p's Backward intervals are obtained and notified of p's existence. This procedure is used for all $O(\log n)$ Forward and Backward intervals of p; therefore, the total cost of updating all such intervals is $O(\log n)$ latency and $O(\log^3 n)$ messages.

After step 4 of $JOIN$, p knows all peers in $Center(p)$. As stated previously, with high probability, the shortest arc length between two peers is $\Theta(1/n^2)$. Thus with high probability, there are no peers in the interval $[p, p+1/n^3]$. In other words, the interval $Forward(p,i)$ for i equal to $\log m - 3\log n$ contains a superset of all the peers in any interval $Forward(p,i)$ for $i < \log m - 3\log n$. A similar argument holds for the backward intervals of peer p. Thus, after executing the for loop of the join protocol, p will know all peers in its Center, Forward and Backward intervals.

Let q be any peer with the property that p is in a Center, Forward or Backward interval for q. Then by Lemma 1, q must be in a Center, Forward or Backward interval for p. Thus, step 3, step 7, or step 9 of the algorithm will notify q of p's existence.

Assume, before p joins the network, that the fraction of faulty peers is no more than $1/4 - \epsilon$ and that all peer points are distributed uniformly at random on the unit circle. Then by Lemma 4, w.h.p., all swarms are good and in particular, $S(p')$ is good. Thus in step 1 of $JOIN$, the peer p will be assigned a random ID and so after p joins the network, all peer points will be distributed uniformly at random on the unit circle. □

A.5 The $STABILIZE$ Protocol

Algorithm 5 gives the pseudocode for the $STABILIZE$ algorithm. This algorithm performs similar functionality to the *stabilize* function in Chord. In particular, $STABILIZE$ keeps a peer's pointers up to date in the face of peer failures and changes in the size of the network. Every peer runs $STABILIZE$ periodically. When a peer p invokes $STABILIZE$, it first determines whether its estimate of $(\ln n)/n$ has changed since the last stabilization procedure. If this estimate has not increased (i.e. the network size has not decreased), then p's Center, Forward and Backward intervals have not increased in size. Links to those peers who belonged, prior to stabilization, in p's Center, Forward or Backward intervals but now lie outside the new, possibly shorter, intervals are deleted. If p's estimate of $(\ln n)/n$ has increased (i.e. the network size has decreased), then p must increase the length of its Center, Forward and Backward intervals. It does this by calling $SUCCESSOR$ repeatedly in order to find the peers which fall in these new, larger intervals.

Lemma 10. *$STABILIZE$ has the following properties with high probability.*

- *It has latency $O(\log n)$ and requires $O(\log^4 n)$ messages (this can be reduced to $O(\log^3 n)$ messages in expectation by using Algorithm 5);*
- *After a peer p calls it, p knows all peers in its Center, Forward and Backward intervals.*

Proof. Peer p has $O(\log n)$ Center, Forwards and Backwards intervals and each of these intervals is of length $O((\log n)/n)$. Thus, there are $\Theta(\log n)$ calls to the $SUCCESSOR$ function in step 3 of $STABILIZE$. Each of these calls has $O(\log n)$ latency and requires $O(\log^3 n)$ messages (or $O(\log^2 n)$ messages in expectation using Algorithm 3).

Algorithm 5 STABILIZE(p)

1: **if** $((\ln n)/n$ has not increased) **then**
2: Peer p updates the length of its Center, Forward and Backward intervals (these intervals will be lengthened);
3: Peer p finds all the new peers in these lengthened intervals by repeatedly calling the $SUCCESSOR$ algorithm;
4: **else**
5: Peer p updates the length of its Center, Forward and Backward intervals (these intervals will be shortened or remain the same);
6: Peer p removes those peers with identifiers that no longer fall into these new intervals;
7: **end if**

The fact that p knows all peers in its Center, Forward and Backward intervals after calling $STABILIZE$ follows directly from Lemma 3. □

A.6 Proofs for Section 5: $\Theta(\log^2 n)$ Expected Messages For SUCCESSOR

Lemma 11. *The expected number of messages sent by correct peers in Algorithm 3 is* $\Theta(\log n)$

Proof. For any peer $x \in S_{j-1}$, let M_x be a random variable giving the number of messages sent by x. Note that $E(M_x) = |S_j|/\log n$. Let M be the sum over all correct peers x of M_x. Then by linearity of expectation,

$$E(M) \leq |S_{j-1}| \cdot |S_j|/\log n = \Theta(\log n)$$

□

For a message m, for any swarm S to which m is sent by the above protocol, let $G(S, m)$ be the set of correct peers in S that receive m (after majority filtering over the accepted messages).

Lemma 12. *For any fixed k and fixed $\epsilon > 0$, for C sufficiently large but depending only on ϵ and k, the following is true with probability $1/n^k$. If $|G(S_{j-1}, m)| \geq (2/3+\epsilon)|S_{j-1}|$, then $|G(S_j, m)| \geq (3/4 - \epsilon)|S_j|$.*

Proof. We will show this lemma by showing that there is a set $Y' \subseteq S_j$, $|Y'| \geq (1-\epsilon)|S_j|$, such that for each peer $y' \in Y'$, more than a 2/3-rd's fraction of the peers that y' accepts messages from are peers in G. Since at least 3/4 of the peers in S_j are correct, this implies that $|G(S_j, m)| \geq (3/4 - \epsilon)|S_j|$.

Let B be the set of integers between 0 and $\log n - 1$. We will refer to elements of B as *bins* and for a peer $x \in S_{j-1}$ and $i \in B$, we will say that x *falls in bin* i if $h_1(x) = i \mod \log n$. We will say that a bin is *bad* if the number of peers in G which fall in the bin is no more than a 2/3-fraction of the number of peers in S_{j-1} that fall in the bin. We first show that, for any fixed $\epsilon' > 0$, for C sufficiently large, with high probability, no more than an ϵ' fraction of the bins are bad.

Fix $B' \subseteq B$ where $|B'| = \epsilon' \log n$. Let $N(G, B')$ be a random variable giving the number of peers in G that fall in a bin in B'. Let $N(S_{j-1}, B')$ be a random variable giving

the number of peers in S_{j-1} that fall in a bin in B'. For all bins in B' to be bad, it must be the case that $N(G, B') \leq (2/3)N(S_{j-1}, B')$. Note that $E(N(G, B')) = C(2/3+\epsilon)\epsilon' \log n$ and $E(N(S_{j-1}, B')) = C\epsilon' \log n$. A simple application of Chernoff bounds gives that for any fixed $\delta > 0$ and fixed $k > 0$, for C sufficiently large, with probability at least $1 - 1/n^{k+2}$:

$$N(G, B') \geq (1-\delta)C(2/3+\epsilon)\epsilon' \log n;$$

and

$$N(S_{j-1}, B') \leq (1+\delta)C\epsilon' \log n.$$

This implies that

$$\begin{aligned} N(G, B')/(N(S_{j-1}, B')) &\geq \frac{(1-\delta)(2/3+\epsilon)}{1+\delta}; \\ &> 2/3. \end{aligned}$$

where the last line follows provided that δ is sufficiently small.

For any $B' \subseteq B$, $|B'| = \epsilon' \log n$, let $\xi(B')$ be the event that all bins in Y' are bad. We've shown that for any $k > 0$, for C sufficiently large, $Pr(\xi(Y')) \leq 1/n^{k+2}$. Now let $\xi_1 = \bigcup_{B' \subseteq B, |B'| = \epsilon' \log n} \xi(Y')$. Then by a simple union bound:

$$\begin{aligned} Pr(\xi_1) &\leq \binom{\log n}{\epsilon' \log n} 1/n^{k+2} \\ &\leq 2^{\log n}(1/n^{k+2}) \\ &\leq 1/n^{k+1} \end{aligned}$$

Now say that any peer $y \in S_j$ is *uninformed* if it falls in a bad bin. Let $U(S_j)$ be a random variable giving the number of uninformed peers in S_j. Let ξ_2 be the event that $U(S_j) > \epsilon |S_j|$. We will now upperbound $Pr(\xi_2 | \bar{\xi_1})$. Assume that event ξ_1 does not occur. Then $E(U(S_j)) \leq \epsilon' |S_j|$. Moreover, since $U(S_j)$ is the sum of $|S_j|$ independent indicator random variables (one for each peer in S_j), we can apply Chernoff bounds. They imply that, for any fixed $\delta > 0$ and fixed $k > 0$, for C sufficiently large, with probability at least $1 - 1/n^{k+1}$:

$$U(S_j) \leq (1+\delta)\epsilon' |S_j|.$$

Choosing δ and ϵ' such that $(1+\delta)\epsilon' = \epsilon$ gives that $Pr(\xi_2 | \bar{\xi_1}) \leq 1/n^{k+1}$. Now note that:

$$\begin{aligned} Pr(\xi_2) &\leq Pr(\xi_1 \cup (\xi_2 | \bar{\xi_1})); \\ &\leq 1/n^{k+1} + 1/n^{k+1}; \\ &\leq 1/n^k; \end{aligned}$$

where the last line follows for n sufficiently large. □

The following lemma gives the robustness of Algorithm 3 over all ordered pairs of swarms.

Lemma 13. *For C sufficiently but depending only on k', the following is true with probability at least $1 - 1/n^{k'}$. For* any *ordered pair of swarms S_{j-1} and S_j, if $G(S_{j-1}, m) \geq (17/24)|S_{j-1}|$, then $G(S_j, m) \geq (17/24)|S_j|$.*

Proof. Let S_{j-1} and S_j be fixed, let ϵ be $1/24$ and $k = k' + 2$ in Lemma 12. Then we have that for C sufficiently large, with probability $1/n^{k'+2}$, if $|G(S_{j-1}, m)| \geq (17/24)|S_{j-1}|$ then $|G(S_j, m)| \geq (17/24)|S_j|$. The number of ordered pairs of swarms is n^2. Thus, a union bound establishes the result for any ordered pair with probability $1/n^{k'}$. □

Lemma 6. *For C sufficiently large but depending only on k', then the following is true with probability at least $1 - 1/n^{k'}$:*

- *All calls to SUCCESSOR succeed.*
- *All calls to SUCCESSOR send $\Theta(\log^2 n)$ messages in expectation.*

Proof. Consider some arbitrary call to SUCCESSOR described by the sequence: $p, S_1, S_2, S_3, \ldots, S_l$. Here p sends the request r to swarm $S_1 = S(p)$, S_1 sends the request to S_2 via Algorithm 3, S_2 sends the request to S_3 via Algorithm 3 and so on. Since p sends the request r to all peers in S_1, $G(S_1, r) \geq (3/4)|S_1|$. Lemma 13 and induction give that $G(S_l, r) \geq (17/24)|S_l|$. Thus $(17/24)|S_l|$ peers in S_l receive the request r and fetch the appropriate data item, d. Thus $G(S_l, d) \geq (17/24)|S_l|$. The data item is now sent back along the path $S_l, S_{l-1}, S_{l-2}, \ldots, S_1, p$. Again by Lemma 13 and induction we can see that $G(S_1, d) \geq (17/24)S_1$. This implies that when p does majority filtering on the messages it receives from S_1, that it will receive the correct message d. Finally, note that since $l = O(\log n)$, by Lemma 11, the expected number of messages sent is $\Theta(\log^2 n)$. □

A.7 Proofs for Section 6: Expected Constant Factor Increase in Number of Bits

Lemma 14. *For any fixed k, for C sufficiently large but depending only on k, the following is true with probability at least $1 - 1/n^k$. For all swarms S, if $|G(S)| \geq |S|/2$ and all peers in S send their pieces to some peer x, then x will be able to reconstruct the message m.*

Proof. We first fix a swarm S and calculate the probability that the statement of the lemma is not true. For any set of correct peers, X, let $U(X) = \{i | h_1(x) = i \mod \log n, \text{for some peer } x \in X\}$. Let S' be some fixed subset of correct peers in S such that $|S'| \geq |S|/2$ and all peers in S' have their pieces. Let P' be some fixed subset of the set of $\log n$ pieces ($\{e_0, e_1, \ldots, e_{\log n - 1}\}$), such that $|P'| > (15/16) \log n$. Let $\xi(S', P')$ be the probability that no peer in X' has a piece in P'. This is equivalent to a balls and bins problem where there are $|S'|$ balls and $\log n$ bins and we are asking the probability that none of the $|S'|$ balls fall in a fixed set of $|P'|$ of the bins. Thus,

$$\begin{aligned} Pr(\xi(X', P')) &\leq (1/16)^{|S'|} \\ Pr(\xi(X', P')) &\leq 1/2^{2|S|} \end{aligned}$$

Now let $\xi(S)$ be the event that for any subsets S' and P', the event $\xi(S', P')$ occurs. Then we have:

$$
\begin{aligned}
Pr(\xi(S)) &= Pr(\bigcup_{S',P'} \xi(S',P')) \\
&\leq \sum_{S',P'} Pr(\xi(S',P')) \\
&= \binom{|S|}{|S|/2}\binom{\log n}{(15/16)\log n} 1/2^{2|S|} \\
&\leq 2^{|S|}2^{\log n}(1/2^{2|S|}) \\
&\leq 1/2^{|S|-\log n} \\
&\leq 1/n^{C-1}.
\end{aligned}
$$

Where the last line follows since $|S| \geq C \log n$. Now, a simple union bound over all n of the swarms then gives that the probability that the statement in the lemma fails for *any* swarm is no more than $1/n^{C-2}$. Choosing C sufficiently large makes this probability no more than $1/n^k$ for any k. □

We will refer to one iteration of the loop in Algorithm 4 as a round.

Lemma 15. *For any fixed k, for C sufficiently large but depending only on k, the following is true with probability $1 - 1/n^k$ for all pairs of swarms, S_{j-1} and S_j. If $|G(S_{j-1})| \geq 1/2|S_{j-1}|$ before Algorithm 4 starts, then $|G(S_{j-1})| \geq 1/2|S_j|$ after termination. Further, if all peers in S_{j-1} know all of the fingerprints before Algorithm 4 starts, then all peers in S_j will know all the fingerprints after termination. Algorithm 4 will:*

- *Terminate in $O(1)$ rounds in expectation;*
- *Require correct peers to send $O(\log^3 n)$ messages in expectation;*
- *Require correct peers to send $O(|m| + \log^4 n)$ bits to be sent in expectation.*

Proof. Since swarm S_{j-1} is good and the peers in S_j do majority filtering in Step 2, we know that if all peers in S_{j-1} know all of the fingerprints before Algorithm 4 starts, then all peers in S_j will know all the fingerprints after termination.

If $|G(S_{j-1})| \geq 1/2|S_{j-1}|$, then by Lemma 14, all peers in the set P which are sent the message pieces in Step 5 will be able to reconstruct the message m. Since 3/4 of the peers in S_j are correct, the probability that no peer in P is correct is no more than $(1 - 1/\log n)^{(3/4)C \log n} \leq e^{-(3/4)C}$. Thus with constant probability, some peer in the set P is correct. If this is the case, then *all* peers in S_j will receive their correct piece of the message. This implies that the algorithm will terminate in that round with $|G(S_j)| \geq 1/2|S_j|$. This implies that Algorithm 4 will terminate in an expected constant number of rounds.

We next establish correctness. Consider the situation where no peer in P is correct. There are then two possible cases. First is the case less than $1/2|S_j|$ peers in S_j are sent their pieces in Step 5. In this case, no peer in Step 2 will receive at least $(3/4)|S_j|$ messages saying that pieces were received. Thus all correct peers will set their resend bits to 1 in this step and so the consensus will be to request a resend. This implies that the algorithm will continue for another round. The second case is that faulty peers in P send pieces to at least $1/2|S_j|$ peers in S_j. If this is the case, then it's safe for the algorithm to terminate.

We now compute the resource costs. Previous to the first round, the fingerprints are sent which requires $O(\log^2 n)$ messages and $O(\log^4 n)$ bits. The expected size of the set P is $O(1)$. Thus, in each round of the algorithm, the total number of messages sent is $O(\log^2 n)$ and the expected total number of bits sent is $O(|m| + \log^3 n)$. The expected resource costs of the entire algorithm then follow directly from the fact that there are $O(1)$ rounds in expectation. □

For any swarm S, we will now let $G(S) = \{x \in S | h_1(x) = i \mod \log n$ and x is correct and has piece $e_i\}$.

Lemma 7. *For any fixed k, for C sufficiently large but depending only on k, the following is true with probability $1 - 1/n^k$, if we run Algorithm 4 with $SEND\_MESSAGE$:*

- *All calls to $SEND\_MESSAGE$ succeed.*
- *All calls to $SEND\_MESSAGE$ have latency $O(\log n)$ in expectation.*
- *All calls to $SEND\_MESSAGE$ require $O(\log^4 n)$ messages to be sent in expectation.*
- *All calls to $SEND\_MESSAGE$ require $O(|m| \log n + \log^5 n)$ bits to be sent in expectation.*

Proof. Consider a message m which is sent along a path described by the sequence: $p, S_1, S_2, S_3, \ldots, S_l, q$. Here peer p sends all pieces of m and fingerprints of these pieces to $S_1 = S(p)$, S_2 sends the pieces and fingerprints to S_2 via Algorithm 4, S_2 sends the pieces and fingerprints to S_3 via Algorithm 4 and so on until finally all peers in swarm S_l sends all their pieces and fingerprints to peer q. Since, $|G(S_1)| \geq 1/2|S_1|$, Lemma 15 and induction give that $|G(S_l)| \geq 1/2|S_j|$ and that all peers in S_l have all the fingerprints of these pieces. This implies that when peers in S_l send their pieces and fingerprints to q, q will have enough information to reconstruct the message m. □

B Handling Different Estimates of $\ln n$ and $(\ln n)/n$

Throughout the previous sections, we have simplified our protocols by assuming that peers know the values $\ln n$ and $(\ln n)/n$ exactly. We now show how to remove this assumption. In particular, we describe how a peer in the enhanced network successfully obtains estimates of $\ln n$ and $(\ln n)/n$ and uses these values in the protocols SUCCESSOR, JOIN, and STABILIZE. Furthermore, we describe how Algorithm 3 and Algorithm 4 can be modified to work when each peer has an estimate of $\log n$.

B.1 Estimates t_p, l_p and h_p

In our DHT, every peer maintains an estimate t_p of $\ln n$, a low estimate l_p of $(\ln n)/n$, and a high estimate h_p of $(\ln n)/n$. In this section we show how a peer can 1) obtain t_p and 2) obtain good low and high estimates of $(\ln n)/n$, l_p and h_p. Many of the techniques we use in this section are similar to those in [13].

B.1.1 Obtaining t_p

Without Byzantine faults, a good estimate of $\ln n$ can be obtained by the methods described in [15, 13, 24]. We demonstrate that this method still provides a good estimate even under our adversarial model. Let $nhbr(p)$ denote the closest clockwise peer to p of which p is aware[5].

Lemma 16. *Let p be a peer in the enhanced DHT and let the fraction of Byzantine peers in the network be f_T. Then, with high probability:*

$$(1/2)\ln n - 0.144 \leq \ln \tfrac{1}{d(p,nhbr(p))} \leq 3\ln n$$

Proof. Let p be a peer in the enhanced DHT and let the fraction of Byzantine peers in the network be f_T. If p is aware of all Byzantine peers, then by [13], for sufficiently large n, then we have:

$$(1/2)\ln n \leq \ln(\tfrac{1}{d(p,nhbr(p)}) \leq (3\ln n)$$

Conversely, if p is aware of none of the Byzantine peers, then:

$$(1/2)\ln((1-f_T)n) \leq \ln(\tfrac{1}{d(p,nhbr(p)}) \leq 3\ln((1-f_T)n))$$

Taking the smallest lower bound and the largest upper bound, we have that:

$$(1/2)\ln(1-f_T) + (1/2)\ln n \leq \ln(\tfrac{1}{d(p,nhbr(p)}) \leq 3\ln n$$

Since $f_T \leq 1/4$ we have:

$$(1/2)\ln n - 0.144 \leq \ln(\tfrac{1}{d(p,nhbr(p)}) \leq 3\ln n$$

□

In order to achieve an estimate of $\ln n$, each peer p sets $t_p = \ln(\frac{1}{d(p,nhbr(p)})$.

B.1.2 Obtaining l_p and h_p

Here we show how a peer p can obtain low and high estimates, l_p and h_p, of $(\ln n)/n$. Let $len(I)$ denote the length of an interval I on the unit circle. The following lemma is due to King and Saia [13]; the proof requires some modifications to account for Byzantine behavior and is presented here for completeness.

Lemma 17. *Let $\alpha_1, \alpha_2, \epsilon$ be positive constants with $\alpha_1 < \alpha_2$ and $0 \leq \epsilon \leq 1$. Let C be a positive constant depending only on $\alpha_1, \alpha_2, \epsilon$ and k. Then for n sufficiently large, with probability $1 - n^{-k}$, the following is true:*

- *Let I be an interval anchored at some peer p on the circumference of the unit circle. If the number of correct peers I contains is greater than $(C\alpha_1)\log n$ and the number of total peers I contains is less than $(C\alpha_2)\log n$. Then, $C(1-\epsilon)\alpha_1(\log n/n) \leq len(I) \leq C(1+\epsilon)\alpha_2(\log n/n)$.*

[5] A faulty peer q may actually be the closest clockwise neighbor of p. However, q may trick p into believing q is dead; consequently, p would not keep a link to q.

Proof. Fix some peer point p, where p is a correct peer, on the unit circle. Let I_s be the interval starting at p and extending clockwise for a distance of $C(1-\epsilon)\alpha_1(\log n/n)$. Let I_l be the interval starting at p and extending clockwise for a distance of $C(1+\epsilon)\alpha_2(\log n/n)$. We will now show that with high probability, I_s contains less than or equal to $C\alpha_1 \log n$ other correct peer points and I_l contains greater than or equal to $C\alpha_2 \log n$ other correct peer points[6].

Let X_s be a random variable giving the number of correct peer points other than p that fall into the interval I_s. Note that a single correct peer point falls in the interval I_s with probability $C(1-\epsilon)\alpha_1(\log n)/n$, so by linearity of expectation:

$$E(X_s) = C(1-\epsilon)\alpha_1((1-f_T)n-1)\log n/n$$

Chernoff [16] bounds tell us that for any δ such that $0 \leq \delta \leq 1$:

$$P(X_s > (1+\delta)E(X_s)) < e^{-\frac{\delta^2 E(X_s)}{3}}$$

Setting $\delta = \epsilon/(1-\epsilon)$ ensures that:

$$\begin{aligned}(1+\delta)E(X_s) &= C\alpha_1(\log n)((1-f_T)n-1)/n \\ &\leq C\alpha_1 \log n\end{aligned}$$

Therefore:

$$\begin{aligned}P(X_s > C\alpha_1 \log n) &< e^{-1\frac{C\epsilon^2\alpha_1((1-f_T)n-1)\log n}{3n(1-\epsilon)}} \\ &\leq e^{-\frac{C\epsilon^2\alpha_1 \log n}{12(1-\epsilon)}} \\ &\leq e^{-\frac{C\epsilon^2\alpha_1 \log n}{12}}\end{aligned}$$

where the second line follows if we assume that $n \geq 2$ (since then $((1-f_T)n-1)/n \geq 1/4$ since $f_T \leq 1/4$) and the third line follows from our assumption that $0 \leq \epsilon \leq 1$.

Now let X_l be a random variable giving the number of correct peer points other than p that fall in the interval I_l. A single correct peer point falls in the interval X_l with probability $C(1+\epsilon)\alpha_2(\log n/n)$, so by linearity of expectation:

$$E(X_l) = C(1+\epsilon)\alpha_2((1-f_T)n-1)\log n/n$$

Chernoff bounds tell us that for any δ, $0 \leq \delta \leq 1$:

$$P(X_l < (1-\delta)E(X_l)) < e^{-\frac{\delta^2 E(X_l)}{2}}$$

We want to choose δ such that $(1-\delta)E(X_l) \geq C\alpha_2 \log n$. Assume that $((1-f_T)n-1)/n \geq \gamma$ for some value $\gamma < 1$. Then we know that $E(X_l) \geq C\gamma(1+\epsilon)\alpha_2 \log n$. Thus to ensure that $(1-\delta)E(X_l) \geq C\alpha_2 \log n$, it suffices if $(1-\delta) \geq \frac{1}{\gamma(1+\epsilon)}$. In other words, we need $\delta \leq 1 - \frac{1}{\gamma(1+\epsilon)}$. To use Chernoff bounds, we have the additional constraint that $0 \leq \delta \leq 1$. Therefore, it must be the case that $\frac{1}{\gamma(1+\epsilon)} < 1$. Choosing $\gamma = \frac{1+\epsilon/2}{1+\epsilon}$ satisfies

[6]We are counting *correct* peers since p may be unaware of some or all of the Byzantine peers and, therefore, cannot count them to obtain an estimate of $(\ln n)/n$.

all of these constraints and requires that $\delta \leq \frac{\epsilon}{2+\epsilon}$. Setting $\delta = \frac{\epsilon}{2+\epsilon}$ (and assuming that $n > \frac{(1+\epsilon)}{\epsilon/2 - f_T(1-\epsilon)}$), we have that:

$$\begin{aligned} P(X_l < C\alpha_2 \log n) &< e^{-\frac{C\alpha_2((1-f_T)n-1)\epsilon^2(1+\epsilon)\log n}{2n(2+\epsilon)^2}} \\ &\leq e^{-\frac{C\alpha_2\epsilon^2(1+\epsilon)\log n}{8(2+\epsilon)^2}} \\ &\leq e^{-\frac{C\alpha_2\epsilon^2\log n}{72}} \end{aligned}$$

where the second line follows if we assume that $n \geq 2$ (since $((1-f_T)n-1)/n \geq 1/4$) and the third line follows by our assumption that $0 \leq \epsilon \leq 1$.

Now for the peer p, consider any anchored interval I that has p as its anchor point. Say that I is *small* if it has length less than or equal to $C(1-\epsilon)\alpha_1(\log n/n)$, and *large* if it has length greater than or equal to $C(1+\epsilon)\alpha_2(\log n/n)$. The bad event for p is that either 1) I is small and I contains greater than $C\alpha_2 \log n$ correct peer points other than p. Let ξ_p be the bad event for the peer p. Then, by a simple union bound, we can say that:

$$\begin{aligned} P(\xi_p) &\leq e^{-\frac{C\epsilon^2\alpha_1\log n}{12}} + e^{-\frac{C\alpha_2\epsilon^2\log n}{72}} \\ &\leq 2e^{-\frac{C\epsilon^2\alpha_1\log n}{72}} \end{aligned}$$

Now let ξ be the event that for any peer p, there exists an interval I anchored at p such that 1) I is small and I contains greater than $C\alpha_1 \log n$ correct peer points other than p or 2) I is large and I contains less than $C\alpha_2 \log n$ peer points other than p. In other words, ξ is the event that the statement of the lemma fails. Again by a simple union bound, we can say that:

$$\begin{aligned} P(\xi) &\leq 2ne^{-\frac{C\epsilon^2\alpha_1\log n}{72}} \\ &= 2ne^{-\frac{C\epsilon^2\alpha_1\ln n}{72\ln 2}} \\ &= 2n^{1-\frac{C\epsilon^2\alpha}{72\ln 2}} \end{aligned}$$

The last equation can be made arbitrarily small for C chosen large enough. □

Lemma 18. *Let C be a sufficiently large positive constant depending only on k, ϵ, and ϵ_0. With probability at least $1 - n^{-k}$, every peer p can obtain low and high estimates of $(\ln n)/n$, l_p and h_p, such that:*

- $(C \ln n)/n \leq l_p \leq h_p$.
- *For any two peers p and q, $l_q \leq h_p$ and $l_p \leq h_q$.*
- *Each interval I with $len(I) \geq (C \ln n)$ contains no more than $1/4(1-\epsilon)C \ln n$ faulty peers.*

Proof. From Lemma 17, p can obtain an estimate est_p of $(\ln n)/n$ such that $(C_1 \ln n)/n \leq est_p \leq (C_2 \ln n)/n$ for constants $1 \leq C_1 \leq C_2$. Let C be a positive constant as in Lemma 4 (depending on k, ϵ, and ϵ_0) so that an interval I with $len(I) = (C \ln n)/n$ is of sufficient length to guarantee, whp, that no more than $1/4(1-\epsilon)C \ln n$ of the peers contained within are faulty. Now p can set $l_p = C/C_1 est_p$ and $h_p = CC_2/C_1 est_p$ which satisfies the claims. □

B.2 Links Required

In this section, we reaffirm certain invariants about the links each peer maintains in our protocol. We redefine the swarm intervals as:

- $S(x) = [x, x + h_x]$ for any point x.
- $Forward(p, i) = [p + \frac{2^i}{m} - h_p, p + \frac{2^i}{m} + h_p]$ for any peer p.
- $Backward(p, i) = [p - \frac{2^i}{m} - l_p, p - \frac{2^i}{m} + l_p]$ for any peer p.
- $Center(p) = [p - 2h_p, p + 2h_p]$ for any peer p.

Lemma 19. *Let x be a point on the unit circle. Then, with probability at least $1 - n^{-k}$, $S(x)$ contains no more than an $1/4(1 - \epsilon)$-fraction of faulty peers; that is, $S(x)$ is good.*

Proof. This follows from the new definition of a swarm and an almost identical argument to what is provided by Lemma 8, Lemma 9, and Lemma 4. □

Lemma 20. *This is the analog to Lemma 1. Let p and q be any two correct peers. Then, the following are true:*

- *p maintains links to all peers in the interval $[p - 2C \ln n/n, p + 2C \ln n/n]$*
- *If p is in the interval $[q + 2^i/m - C \ln n/n, q + 2^i/m + C \ln n/n]$ for some fixed i where $1 \leq i \leq \log m - 1$, then:*
  - *q is in the interval $[p - 2^i/m - C \ln n/n, p - 2^i/m + C \ln n/n]$*
  - *q maintains a link to p and vice versa.*
- *If p is in the interval $[q - 2^i/m - C \ln n/n, q - 2^i/m + C \ln n/n]$ for some fixed i where $1 \leq i \leq \log m - 1$, then:*
  - *q is in the interval $[p + 2^i/m - C \ln n/n, p + 2^i/m + C \ln n/n]$*
  - *q maintains a link to p and vice versa.*

Proof. Note that $(C \ln n)/n \leq l_p \leq h_p$ and the same for l_q and h_q. Therefore, this follows directly from the new definitions of Center, Forward, and Backward intervals of p and q. □

Lemma 21. *With high probability, the number of peers any peer links to is $\Theta(log^2 n)$*

Proof. This proof is virtually identical to Lemma 2. □

B.3 Successor Protocol

The pseudocode for SUCCESSOR using estimates of l_p and h_p is given in Algorithm 6. A crucial point about SUCCESSOR, as well as JOIN and STABILIZE, is that not all correct peers in a swarm may choose to partake in the protocol due to differing estimates of $(\ln n)/n$. However, we can always guarantee, with high probability, that all correct peers in some sub-interval of the swarm, of length greater than or equal to $(C \ln n)/n$, are acting as the protocol directs. This guarantee is what allows our protocols to function successfully.

If peer p desires the content associated with key k, it uses its high estimate to alert those peers in $[p, p+h_p]$ of its request. Each peer q receiving the alert decides whether it belongs to $S(p)$ by determining whether p lies in a counterclockwise interval of length l_q from q. If so, q forwards the request as originally described to the interval $[x-h_p, x+h_p]$ where $x = next(p,k)$. A peer q along the path traversed by SUCCESSOR only forwards the request once it has received it from a majority of peers in one of its backward intervals now defined using l_p. A peer q who receives the same request from a majority of peers in one of its backward intervals, forwards the request to the interval defined by $x' = next(x,k)$ using h_q; that is, the interval $[x', x'+h_q]$. The SUCCESSOR operation ends once $d(x,k) \leq l_p$ at which point the set of all identifiers to all peers in $S(k)$ are sent back to p along the search path.

Algorithm 6 SUCCESSOR(k)

1: p sends a request for k to all peers in $[p, p+h_p]$;
2: $S \leftarrow$ set of all peers in $[p, p+h_p]$;
3: $x \leftarrow$ identifier of p;
4: Every peer q which received the request for k from a majority of peers in $[x, x+l_q]$ do the following:
5: **if** $(d(x,k) \leq l_q)$ **then**
6: q sends back all identifiers in $S(k)$ to all peers that sent q the request (these identifiers are recursively sent back along the same path to p).
7: **else**
8: $x \leftarrow next(x,k)$;
9: q sends request for k to all peers in $[x, x+h_q]$;
10: **end if**

Lemma 22. *Peers possess the required links to correctly perform $SUCCESSOR(k)$ for any key $k \in (0,1]$ and any peer p initiating the request.*

Proof. Here we show that all peers in the network possess the links required to carry out the SUCCESSOR protocol; in Lemma 23, we will show correctness.

The initial step in the protocol requires p be able to alert its swarm $S(p)$ to p's request. This requirement is satisfied because p maintains links to all peers in $Center(p) = [p-2h_p, p+2h_p]$ and so p maintains links to all peers in $[p, p+h_p]$. Each peer q in the interval $[p, p+h_p]$ must then be able to decide whether it belongs to $S(p)$. Since q also maintains links to all peers in $Center(q)$, q can determine whether or not p lies in the interval $[q-q_l, q]$. If so, q agrees that it belongs to $S(p)$ and forwards the request; otherwise, q ignores the request.

By the protocol, every peer q in $S(x)$ must be able to send the request to all peers in $S(next(x,k)) = [next(x,k), next(x,k)+h_q]$. In other words, for any i between 1 and $\log m - 1$, all peers in $[x, x+h_q]$ must maintain links to the set F of all peers in $[x+2^i/m, x+2^i/m+h_q]$. However, each peer q maintains Forward intervals and, therefore, knows the set F' of peers in $[q+2^i/m-h_q, q+2^i+h_q]$. Since $F \subseteq F'$, all necessary forward links exist for q.

For the purposes of majority filtering on incoming requests, each peer q in the interval $[next(x,k), next(x,k)+h_q]$ needs to know the set B of all peers in the interval $[x, x+(C\ln n)/n]$. However, each peer q tracks Backward intervals and, therefore, knows the set B' of peers in $[x-2^i/m-l_q, x-2^i/m+l_q]$. Since $B \subseteq B'$, all peers have the required backward links.

Finally, the *while* loop of this protocol terminates when $d(x,k) \leq l_p$. Assume this happens once the request is received by peers in $S(x)$. Each peer q' in $S(x)$ needs to know all peers in $S(k) = [k, k+l_q]$. This requirement is satisfied since each peer q tracks a Center interval and, therefore, knows all peers in the interval $[x-2h_q, x+2h_q]$. Since $h_q \geq C \ln n/n$, q knows all the peers in $[k, k+l_q]$. Therefore, all peers have the required links to terminate SUCCESSOR and return the identifiers of $S(k)$. □

Lemma 23. *For any key k and peer p, SUCCESSOR(k) always returns pointers to all peers in $S(k)$ when called by p. Moreover, SUCCESSOR has latency $O(\log n)$ and requires $O(\log^3 n)$ messages.*

Proof. We first argue the correctness of SUCCESSOR by showing that at each hop, all peers within a subinterval of size at least $(C \ln n)/n$ are partaking in the protocol. Initially, p sends the request for k to all peers in $S(p) = [p, p+h_p]$. Since $h_p \geq (C \ln n)/n$, Lemma 4 guarantees with high probability that $S(p)$ is good. Every peer $q \in S(p)$ uses its estimate q_l to determine whether it belongs to $S(p)$. Therefore, all good peers in the interval $[p, p + (C \ln n)/n]$ receive the request *and* will forward it to peers in $S(x', k) = [x', x' + h_q]$ where $x' = next(p, k)$. Again, by the definition of high estimates, all peers in the interval $[x', x' + (C \ln n)/n]$ will receive the forwarded request.

Consequently, at each forwarding of the request in Figure 6, each peer q' in the interval $[x', x' + h_q]$ will receive the request from a majority of peers in a sub-interval of $Backward(q, i)$ of size at least $(C \ln n)/n$ for some i. Again, each peer q' in $[x', x' + h_q]$ will decide if it belongs to $S(x')$ and, by Lemma 4, the majority of such peers will be correct. Each peer q' will then majority filter on the request and forward it to the next interval $[next(x', k), next(x', k)+h_{q'}]$ which is again at greater than or equal to $(C \ln n)/n$ in length ; therefore, correctness is ensured inductively.

The argument showing that $SUCCESSOR(k)$ always returns pointers to all peers in $S(k)$ upon termination of the *while* loop is almost identical to that given in Lemma 3 and we do not repeat it here. For the same reason, a proof that the latency is $O(logn)$ is also omitted from this extended abstract. □

As before, an almost identical algorithm can be used to implement the $SEND\_MESSAGE()$ protocol.

B.4 Join

Algorithm 7 provides the new pseudocode for the JOIN protocol. No longer can a swarm $S(p)$ use an exact value $(C \ln n)/n$ in establishing the size of the Forward, Backward, and Center intervals of the joining peer p. Instead, $S(p)$ must calculate l_p and h_p and use those values to establish interval sizes. Moreover, it is inaccurate to speak of all peers in $S(p)$ performing operations since high estimates of peers may differ and, consequently, correct peers may not operate on the same intervals (via sending messages to them or receiving messages from them). Instead, we will show that, with high probability, at least all correct peers in an interval of size $(C \ln n)/n$ are performing the operations needed to facilitate p's entry into the network. Let S_q denote the set of peers in $[q, q + (C \ln n)/n]$ and S_p denote the set of peers in $[p, p + (C \ln n)/n]$.

Lemma 24. *The JOIN protocol has the following properties with high probability:*

- *JOIN has $\Theta(\log n)$ latency and $\Theta(\log^3 n)$ message complexity.*

Algorithm 7 JOIN(p)

1: Peer p contacts some correct peer q which notifies S_q of p's request to join;
2: Peer q initiates a secure computation with peers in S_q of a random number in $(0, 1]$ using the algorithm discussed in Section C;
3: All peers in S_q get pointers to the peers in $Center(p)$, using $O(1)$ calls to the $SUCCESSOR$ algorithm. All peers in S_q send these pointers to p;
4: All peers in S_q calcululate t_p, l_p and h_p. S_q then notifies the peers in $Center(p) = [p - 2h_p, p + 2h_p]$, using the $SEND\_MESSAGE$ algorithm, that p has joined the network - note that p's estimation is used;
5: The peers in $Center(p)$ send data items for all keys k such that $p \in S(k)$ and p then stores copies of these data items;
6: **for** $i = (\log m - 3t_p)$ to $\log m$ **do**
7: Each peer p' in S_p tells all peers in the set S_1 of peers in $[p + 2^i/m, p + 2^i/m + h_q]$ of p's arrival. Each peer p'' in S_1 then tells all peers in the set of peers S_2 in $[p + 2^i/m - h_{q'}, p + 2^i/m]$ of p's arrival. In this way, all peers in $Forward(p, i)$ know about p;
8: In an almost identical fashion to Step 7, all peers in S_p get pointers to the peers in $Forward(p, i)$. All peers in S_p then send these pointers to p;
9: Each peer p' in S_p tells all peers in the set S_3 of peers in $[p - 2^i/m, p - 2^i/m + h_q]$ of p's arrival. Each peers p'' in S_4 then tells each peer in the set S_4 of peers in $[p - 2^i/m - h_{q'}, p - 2^i/m]$ of p's arrival. In this way, all peers in $Backward(p, i)$ know about p;
10: In an almost identical fashion to Step 9, all peers in S_p get pointers to the peers in $Backward(p, i)$. All peers in S_p then send these pointers to p;
11: **end for**

- *After JOIN completes, peer p knows all peers in its Center, Forward and Backward intervals.*
- *Let q be any peer in a subinterval* $[p - (2C \ln n)/n, p + (2C \ln n)/n]$ *of* $Center(p)$*, a subinterval* $[p + \frac{2^i}{m} - (C \ln n)/n, p + \frac{2^i}{m} + (C \ln n)/n]$ *of* $Forward(p, i)$*, or a subinterval* $[p - \frac{2^i}{m} - (C \ln n)/n, p - \frac{2^i}{m} + (C \ln n)/n]$ *of* $Backward(p, i)$ *for* $1 \leq i \leq \log m - 1$*. Then after JOIN completes, q knows about the peer p.*
- *Assume, before p joins the network, that the fraction of faulty peers is no more than* $1/4 - \epsilon$ *and that all peer points are distributed uniformly at random on the unit circle. Then* after *p joins the network, all peer points are distributed uniformly at random on the unit circle.*

Proof. The arguments for the first, second, and fourth claims are virtually identical to those given in the proof given for Lemma 5 and we do not repeat them here. Statement 3, follows due to the fact that S_q alerts $Center(p)$ using $h_p \geq (C \ln n)/n$, that S_p alerts its Forward and Backward intervals using h_p, and Lemma 20. □

B.5 Stabilize

The new $STABILIZE$ protocol given as Algorithm 8 changes only slightly from the pseudocode given in Algorithm 5. Instead of knowing whether $(\ln n)/n$ has changed in size, peer p must recalculate l_p and determine whether this value has changed. If

l_p has decreased in value, then p infers that the network has decreased in size. Then, the Forward, Backward, and Center intervals of p need to be expanded via calls to $SUCCESSOR$. If l_p is unchanged or has increased in value, then p infers that the network has grown. In this case, $STABILIZE$ is less costly since the Forward, Backward, and Center intervals of p need to shrink. This is accomplished by simply removing links to those peers that no longer belong to these updated intervals.

Algorithm 8 STABILIZE(p)

1: **if** (p's estimate of l_p has not increased) **then**
2: Peer p updates the length of its Center, Forward and Backward intervals {these intervals will be lengthened};
3: Peer p finds all the new peers in these lengthened intervals by repeatedly calling the $SUCCESSOR$ algorithm;
4: **else**
5: Peer p updates the length of its Center, Forward and Backward intervals (these intervals will be shortened or remain the same);
6: Peer p removes those peers with identifiers that no longer fall into these new intervals;
7: **end if**

Lemma 25. *$STABILIZE$ has the following properties with high probability.*

- *It has latency $O(\log n)$ and requires $O(\log^4 n)$ messages (this can be reduced to $O(\log^3 n)$ messages in expectation by using Algorithm 5);*
- *After a peer p calls it, p knows all peers in its Center, Forward and Backward intervals.*

Proof. This is virtually identical to the proof given for Lemma 10. □

B.6 Obtaining an Additive Factor Approximation to $\log n$

Previous work [13] shows how, with high probability, each peer in a DHT can estimate $\log n$ to within an arbitrarily close constant factor. We now reaffirm this result for our DHT.

Lemma 26. *Let C_0 be a positive constant. In our enhanced DHT, with high probability, any peer, p, can obtain an estimate u_p of $\log n$ such that:*

$$\log n \leq u_p \leq \log n + C_0$$

Proof. By Lemma 16, p can obtain a value t_p such that:

$$(1/2)\ln n - 0.144 \leq t_p \leq 3\ln n.$$

By Lemma 17, p can use this value t_p, to obtain an estimate est_p of $(\ln n)/n$ such that for constants $1 \leq C_1 \leq C_2$:

$$(C_1 \ln n)/n \leq est_p \leq (C_2 \ln n/n).$$

The following is then true:

$$\frac{(\frac{1}{2}\ln n - 0.144)}{(C_2 \ln n)/n} \leq \frac{t_p}{est_p} \leq \frac{3\ln n}{(C_1 \ln n)/n}$$

This implies that:

$$\log n - C_1 \leq \log \frac{t_p}{est_p} \leq \log n + C_2$$

Thus

$$\log n \leq \log \frac{t_p}{est_p} + C_1 \leq \log n + C_1 + C_2.$$

□

B.7 Different Estimates of $\log n$ for Algorithm 3

We now discuss how to modify the algorithm to handle different estimates of $\log n$. By Lemma 26, for any two peers p and p':

$$\log n \leq l_p \leq u_{p'} \leq \log n + C_0.$$

We then change the first two bullets of our algorithm as follows:

- Each peer $x \in S_{j-1}$ sends a message to peer $y \in S_j$ iff

$$h_1(x) = h_1(y) \mod l_x$$

- Each peer $y \in S_j$ accepts a message from peer $x \in S_{j-1}$ iff

$$h_1(x) = h_1(y) \mod u_y$$

Showing the modified algorithm is correct requires only minor modifications of the proof of Lemma 12. Let B again be the set of bins from 0 to $\log n - 1$. It is straightforward to show that, with high probability, an arbitrarily small fraction of the bins in B are "bad". Then, with high probability, an arbitrarily large fraction of the peers in S_j fall in bins in B which are good. We finally note that the algorithm is robust to minor inconsistencies in the views the peers have as to which peers are in the swarms S_{j-1} and S_j.

B.8 Different Estimates of $\log n$ for Algorithm 4

We now discuss how to modify the Algorithm 4 to handle different estimates of $\log n$. As before, we will assume that each peer p has an estimate with the following property for fixed constant C_0.

$$\log n \leq l_p \leq \log n + C_0.$$

The algorithm is then modified as follows. The peer p which starts sending the message m encodes m into l_p pieces and creates fingerprints for all of these pieces. When it sends the fingerprints to the swarm S_1, it also sends the number l_p to all peers in S_1. The peers in S_1 then use this number l_p as their estimate of $\log n$. When the peers in S_1 send the fingerprints to peers in S_2, all peers in S_1 also send the number l_p to all peers in S_2. In this way, we maintain the invariant that all peers in the swarms m is sent to know and use the estimate, l_p, calculated by p.

We finally note that this algorithm is robust to minor inconsistencies in the views the peers have as to which peers are in the swarms S_{j-1} and S_j.

C Assigning Random Identifiers

In this section, we describe how all correct peers in a good swarm can come to consensus on a random number in $(0, 1]$. Our algorithm for doing this makes use of results in the area of secure multiparty computation for asynchronous networks which we now describe.

A secure multiparty computation protocol allows a set of s players, t of which are adversarially controlled, to compute the value of an agreed upon multi-variate function $\mathcal{F}$, while keeping their local inputs private (see e.g. [6]). In the case of an asynchronous network, the inputs to the computation come from an arbitrary core subset C of the set of players where $|C| \geq s - t$; for the purpose of computing the function, the inputs of all players not in C are assumed to be some default value (say 0). There are several results showing how to achieve secure multiparty computation in an asynchronous network provided that $t \leq s/4$ (see e.g. [2, 21, 18]).

Using these results, it is straightforward to see how all peers in a good swarm can come to consensus on a random number in $(0, 1]$. Each peer first chooses as input a random number in $(0, 1]$. The peers then use the secure computation algorithm to compute the sum, σ, of the inputs from the set C. Each peer then accepts $\sigma \mod 1$ as the random number chosen. Since at least one player in C must be a correct peer, $\sigma \mod 1$ must be a random number in $(0, 1]$.

Srinathan and Rangan [21] give the most resource efficient secure multiparty computation protocol of which we are aware for this problem. In the case where there are $\Theta(\log n)$ peers, no more than $1/4$ of which are faulty, we can compute a random number using $\Theta(\log^3 n)$ messages, with $\Theta(\log n)$ latency using their protocol.