Choosing a Random Peer in Chord

Valerie King * Scott Lewis † Jared Saia †‡
Maxwell Young †

Abstract

We present two new algorithms, *Arc Length* and *Peer Count*, for choosing a peer uniformly at random from the set of all peers in Chord [24]. We show analytically that, in expectation, both algorithms have latency $O(\log n)$ and send $O(\log n)$ messages. Moreover, we show empirically that the average latency and message cost of *Arc Length* is $10.01 \log n$ and that the average latency and message cost of *Peer Count* is $20.02 \log n$. To the best of our knowledge, these two algorithms are the first fully distributed algorithms for choosing a peer uniformly at random from the set of all peers in a Distributed Hash Table (DHT). Our motivation for studying this problem is threefold: to enable data collection by statistically rigorous sampling methods; to provide support for randomized, distributed algorithms over peer-to-peer networks; and to support the creation and maintenance of random links, and thereby offer a simple means of improving fault-tolerance.

Key Words: *Peer-to-peer, Distributed Hash Table, Chord, Randomized Algorithms, Distributed Algorithms, Data Collection, Attack-resistance*

1 Introduction

In this paper, we address the problem of choosing a peer uniformly at random from the set of all peers in Chord [24]. Random sampling is a fundamental statistical operation; a function which chooses a random peer can be used for many types of applications, including the following:

- *Collecting Data:* By randomly sampling peers, we can quickly collect the following types of useful information: peer opinions, e.g., on popular content; physical properties of network nodes, e.g., for measurement studies like [23, 22]; and environmental data, e.g., for sensor networks.

*Department of Computer Science, University of Victoria, P.O. Box 3055, Victoria, BC, Canada V8W 3P6; email: val@cs.uvic.ca

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

‡*Contact Author* Phone: 505-277-5446, Fax: 505-277-6927

- *Providing an Algorithmic Building Block:* An algorithm for randomly sampling a peer can be used as a building block for other distributed algorithms. For example, there are at least two currently published algorithms for peer-to-peer networks which require an algorithm for choosing a random peer. The first algorithm ensures good load-balancing of computational tasks across the peers in a network [10]. The second algorithm provides a scalable solution to the Byzantine agreement problem [14]. While both results critically rely on the existence of an algorithm to choose a random peer, they only suggest heuristics to solve this problem.

- *Making Networks More Robust:* An algorithm that randomly samples peers can be used to create more robust networks. Consider a network where every node has a small number of links to other random nodes. Adding these random links will turn the network into an expander [18]. A network which is an expander is known to be robust in the sense that it will stay well-connected even in the face of a sudden, massive number of adversarial node deletions [18]. An algorithm for choosing a random peer allows for simple creation and maintenance of random links, and thus can provide an extra measure of robustness.

1.1 Problem Statement

A Distributed Hash Table (DHT) is a distributed, scalable indexing scheme for peer-to-peer networks. A DHT is typically used to provide for efficient storage and lookup of large numbers of data items. Many DHTs have been proposed in the literature [1, 2, 3, 24], but one of the most popular is Chord [24] .

We now describe Chord. Chord has a *key space* which is scaled so it is in the range $(0, 1]$. We can think of the key space of Chord as a circle with unit circumference, which we will call the *unit circle*. We assume that n peers participate in Chord and that these peers are mapped to locations on the unit circle which we call *peer points*. The n peer points are assumed to be distributed uniformly at random on the unit circle. In particular, there is a *base hash function* which maps peers, based on their IDs[1], to points on the unit circle and Chord makes the random oracle assumption [7] about this base hash function, i.e. that it maps IDs to essentially random locations on the unit circle.

Chord provides two basic operations: h and $next$. For a point x on the unit circle, $h(x)$ is the peer whose peer point is closest in clockwise distance to x. For a given peer p, $next(p)$ returns the peer whose peer point is closest in clockwise distance to p's peer point. Single applications of h and $next$ have latencies of $\log n$ and 1, respectively, and require $\log n$ and 1, respectively, messages to be sent[2].

Our problem then is to design a scalable, distributed algorithm which chooses a peer uniformly at random from the set of all peers in the Chord. We want this algorithm to use only the basic operations h and $next$ and we want it to be

[1] e.g. IP-addresses

[2] Throughout the paper, we will use log to represent log base 2.

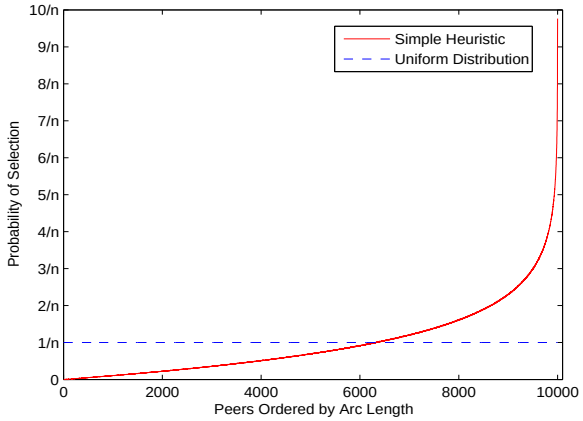


Figure 1: Probability of selecting a peer using the simple heuristic. Note that there is significant bias toward peers that have large counter-clockwise arc lengths. The dashed line is the uniform distribution, which is achieved by the algorithms presented in this paper.

scalable in the sense that latency and bandwidth will be at most polylogarithmic in n.

A simple heuristic for this problem is to choose a random point x on the unit circle and return $h(x)$. While this simple heuristic may be useful when only approximation to uniform sampling is needed, the heuristic can have significant bias as we now show. The probability that a peer p is chosen by this heuristic is proportional to the length of the arc between the peer point for p and the closest counter-clockwise peer point. The lengths of these arcs vary widely. With high probability[3], the longest arc is of length $\Theta(\log n/n)$ [24] and the shortest arc is of length $\Theta(1/n^2)$ [12]. Thus, the peer with the longest arc will be chosen $\Theta(n \log n)$ times more frequently than the peer with the shortest arc. Figure 1 shows empirically that this heuristic has significant bias. This plot represents the results of using the simple heuristic on 100 random DHTs, each consisting of $10,000$ peers. The peers in each DHT are sorted on the x-axis of the plot according to their arc lengths, which are computed as described above. The y-axis gives the fraction of the time each peer was selected over $5,000,000$ executions of the simple heuristic (averaged over trials on each of the 100 DHTs). This plot shows that there is significant bias not only for the peers with minimum and maximum arc length but for many of the other peers as well. To remove this bias, we require a more sophisticated algorithm.

[3]Throughout this paper, we will use the phrase "with high probability" to mean with probability $1 - n^{-c}$ for some fixed constant c greater than 1.

1.2 Our Results

Our main theoretical result is stated in the following theorem which is proven in Section 3.

Theorem 1. *Assume n peers are distributed uniformly at random on the unit circle of Chord. Then with probability $1-3/n$, both* Arc Length *and* Peer Count *have the following properties every time they are called by any peer in Chord.* [4]

- *They choose each peer with probability* exactly $1/n$*;*
- *In expectation, they have latency $O(\log n)$ and send $O(\log n)$ messages;*
- *With high probability, they have latency $O(\log^2 n)$ and send $O(\log^2 n)$ messages.*

Both *Arc Length* and *Peer Count* have the same asymptotic resource costs. However, the hidden constants in these asymptotic bounds can be quite different. For this reason, we turn to empirical analysis to compare the performance of these two algorithms in practice.

Our main empirical results are given in Section 4. In that section, we empirically test both *Arc Length* and *Peer Count* and show that both algorithms perform well in practice. In particular, we show that in practice, for $n \geq 10,000$, the average latency and message cost of a single call to *Arc Length* is $10.01 \log n$ and the average latency and message cost of a single call to *Peer Count* is $20.02 \log n$. This means, for example, that for a DHT containing one million peers, *Arc Length* has latency and message cost less than 220 while *Peer Count* has latency and message cost less than 400.

A preliminary version of *Peer Count* appeared in [12]. In this paper, we present a new simpler version that tightens some of the parameters of the preliminary version in order to improve empirical performance. This paper introduces the algorithm *Arc Length*.

1.3 Related Work

Gkansidis et. al. address the problem of choosing a random peer in a peer-to-peer system [9]. They show that random walks can provide a good approximation to uniform sampling for networks where the gap between the first and second eigenvalues of the transition matrix is constant. Their result only approximates uniform sampling and the closeness of the approximation is impossible to formally state without knowledge of the second eigenvalue of the network. See also Law and Siu [13] who also use random walks to sample peers approximately.

There are several results on adding load-balancing extensions to the basic DHT model. These results seek to more equitably map the function h across the peers. See [24] for a technique involving *virtual nodes* in which each peer

[4] In particular, for any base hash function of Chord, with probability $1-3/n$, our algorithms have these properties every time they are called by any peer.

maps to $O(\log n)$ peer points on the unit circle and [8, 6, 20, 11] for other techniques. Generally these techniques work by dynamically "reassigning" hash space among the peers to ensure that no peer is ever responsible for too large a portion of the unit circle.

We have assumed a standard DHT which has no load-balancing extensions. We make this assumption for two reasons. First, we would like our protocols to be applicable for a wide range of DHTs and there is currently no consensus about the best way to add load-balancing extensions to a DHT. Also the results we have for the basic Chord model can be easily adapted to a DHT which has load-balancing extensions. Second, we want our protocol to work on DHTs which are robust to malicious faults such as [21, 5]. Such DHTs provide the same functionality as Chord and can robustly provide the h and $next$ operations even in the presence of large numbers of malicious faults. Thus, our algorithms will work in the presence of malicious faults when they are run on these DHTs. This is of critical importance if we will be using our algorithms for choosing random peers as subroutines in other attack-resistant algorithms for a DHT (e.g. Byzantine agreement [14]). Unfortunately, we are not aware of any DHTs with load-balancing extensions which are provably robust to malicious faults.

1.4 Notation

For any two points x and y on the unit circle, we let $d(x, y)$ be the distance from x to y traveling clockwise along the unit circle i.e. if $y \geq x$, then $d(x, y) = y - x$ else $d(x, y) = (1 - x) + y$. For points x and y on the unit circle, we will use $(x, y]$ to refer to the interval on the perimeter of the unit circle traveling clockwise from x to y. For brevity, we will frequently use the word interval to mean "interval on the perimeter of the unit circle". For an interval I, we will let $len(I)$ denote the length of I and will let $num(x, y)$ denote the number of peer points in I.

For a given peer, p, we will use p interchangeably to refer both to the peer itself and to the peer point for p. The exact meaning will be clear from context. For any peer p, we note that k applications of $next$ returns the k^{th} next peer in the clockwise ordering around the circle from p and is denoted $next^{(k)}$.

The rest of this paper is laid out as follows. In Section 2, we give our two algorithms for choosing a random peer. We analyze these algorithms and give proofs of correctness in Section 3. In Section 4, we describe our empirical results for these two algorithms. Section 5 describes a possible extension of *Arc Length* for unstructured networks and Section 6 concludes and gives directions for future work.

2 Algorithms

We now present the algorithms *Peer Count* and *Arc Length*. *Peer Count* depends on the ability of each peer p to independently determine a number t_p and a length $dmin_p$ such that with high probability, no interval containing t_p peers has length less than $dmin_p$. *Arc Length* depends on the ability of each peer p

to independently determine a length d_p and a number $tmax_p$ so that with high probability, no interval of length d_p contains more than $tmax_p$ peers. In the next subsection, we show how these parameters can be chosen such that t_p and $tmax_p$ are both $\Theta(\log n)$ and $dmin_p$ and d_p are $\Theta(\log n/n)$.

Both algorithms use $O(\log n)$ calls to *next* in expectation and an expected constant number of calls to h for suitably chosen parameters. We first describe the algorithms and then describe the procedures for choosing parameters.

2.1 Algorithm *Peer Count*

The algorithm *Peer Count* is presented formally in Figure 2. A peer p initially calls $FindParametersI$ to determine values for $dmin_p$ and t_p and sets λ to $dmin_p/t_p$. Then the algorithm enters a loop in which it selects a random number r from $(0, 1]$. It moves clockwise around the circle to the next peer until a peer p' is encountered such that $d(r, p') < \lambda num(r, p')$ or t_p peers have been examined. If such a peer is found, it is returned; otherwise, the loop is repeated. One execution of a loop is referred to as a *round*.

The high level intuition for the correctness of algorithm *Peer Count* is as follows. If the parameters $dmin_p$ and t_p are set correctly, then λ will be $\theta(1/n)$. Algorithm *Peer Count* will associate each peer with exactly λ length of "real estate" on the unit circle. If the random value r falls in the real estate belonging to peer p, then p will be chosen by *Peer Count*. Peers with short arc lengths will get extra real estate from peers with longer arc lengths. Thus, the real estate associated with a particular peer need not be contiguous on the unit circle. We can show using Chernoff bounds that any interval containing t_p contiguous peers (i.e. an interval considered by the algorithm) will have length large enough to assign λ real estate to each peer in the interval. The value T in algorithm *Peer Count* is used to partition up the length of such an interval so that each peer has exactly λ real estate assigned to it. The formal proof of correctness of algorithm *Peer Count* is presented in Section 3.

2.2 Algorithm *Arc Length*

The algorithm *Arc Length* is presented formally in Figure 3 and we give an overview here. A peer p calls $FindParametersII$ to select parameters d_p and $tmax_p$ such that with high probability, no interval of length d_p contains more than $tmax_p$ peers. Then the algorithm enters a loop in which it selects a random number r from $(0, 1]$ and a random integer x in $[1, tmax_p]$. The algorithm then moves clockwise around the circle to the next peer until it has examined x peers or it has moved a distance greater than d_p from the point r. If the algorithm finds a peer p' such that 1) p' is the x^{th} peer it has encountered moving clockwise from r and 2) $d(r, p') \leq d_p$, then p' is returned; otherwise the loop is repeated. One execution of a loop is referred to as a *round*.

The high level intuition for the correctness of algorithm *Arc Length* is as follows. In one round of the algorithm, some interval I starting at point r and of length d_p will be considered. There will be some number, n', of peers in interval

1. $t_p, dmin_p \leftarrow$ FindParametersI();
2. $\lambda \leftarrow dmin_p/t_p$;
3. While TRUE do :
4. $\quad r \leftarrow$ random number in $(0, 1]$;
5. $\quad first \leftarrow h(r)$; $T \leftarrow d(r, first) - \lambda$;
6. $\quad$ Repeat $t_p - 1$ times or until $T < 0$:
7. $\quad\quad T \leftarrow T + d(first, next(first)) - \lambda$;
8. $\quad\quad first \leftarrow next(first)$.
9. $\quad$ If $T < 0$ return $first$;

Figure 2: Algorithm *Peer Count*

I. If $FindParametersII$ works correctly, n' will be less than $tmax_p$. Thus, Algorithm *Arc Length* selects each peer in interval I with probability exactly $1/tmax_p$. Since each peer has probability d_p of being in the interval considered by *Arc Length* in a round, it means that each peer has probability exactly $d_p/tmax_p$ of being selected in a given round. The formal proof of correctness of algorithm *Arc Length* is presented in Section 3.

2.3 Choosing parameters

Here we describe the procedures FindParametersI and FindParametersII. These procedures use constants c_1, c_2, c_3, c_4, which will be tuned to minimize latency and ensure correctness. Both procedures use only estimates of $\ln n$ and $(\ln n)/n$ since the size n of the networks is not known to each peer.

For sufficiently large n, with probability $1 - 1/n$, a constant approximation of $\ln n$ is given by the distance from a peer to its nearest clockwise neighbor, as in [15]. In Figure 4, we generalize this approach: Procedure 1 gets its estimate based on the distance between p and its c_1^{th} closest clockwise neighbor.

An algorithm for estimating $(\ln n)/n$ is given in [16, 21]. For sufficiently large n, with probability $1 - 1/n$, the distance spanned by any $\Theta(\ln n)$ peers is $\Theta(\ln n/n)$. In Figure 5, Procedure 2 generalizes the algorithm from [16, 21] by introducing the constant c_2.

The procedures FindParametersI and FindParametersII are given in Figure 6 and Figure 7, respectively. These procedures first get estimates of $\ln n$ and $(\ln n)/n$ and then compute t_p and $dmin_p$ (respectively, d_p and $tmax_p$).

1. $d_p, tmax_p \leftarrow$ FindParametersII();
2. While TRUE do :
3. $\quad r \leftarrow$ random number in $(0, 1]$;
4. $\quad x \leftarrow$ random integer in $[1, tmax_p]$;
5. $\quad i \leftarrow 1$; $p' \leftarrow h(r)$;
6. $\quad$ While $i < x$ and $d(r, p') \leq d_p$:
7. $\quad\quad i \leftarrow i + 1$; $p' \leftarrow next(p')$;
8. $\quad$ If $d(r, p') \leq d_p$ return p';

Figure 3: Algorithm *Arc Length*

1. $p \leftarrow$ id of self;
2. $\hat{n} \leftarrow c_1/[d(p, next^{(c_1)}(p))]$;
3. Return $\ln \hat{n}$;

Figure 4: Procedure 1: Estimating $\ln n$

3 Analysis

In this section, we prove Theorem 1. The proof of Theorem 1 will make use of Lemmas 2, 6, 8, and 12 which we give in this section.

3.1 Analysis of FindParametersI

In this section, we will prove the following lemma about the procedure FindParametersI.

Lemma 2. *Assume n peers are distributed uniformly at random on the unit circle of a DHT. Then there exist settings for the constants c_1, c_2, c_3, c_4 in the procedure FindParametersI which ensure the following with probability at least $1 - 3/n$. For every peer p, t_p and $dmin_p$ are chosen such that:*

- *Every interval containing t_p peers has length at least $dmin_p$;*
- *$t_p = \Theta(\ln n)$ and $dmin_p = \Theta((\ln n)/n)$.*

1. $s \leftarrow$ estimate of $\ln n$, via Procedure 1;
2. Return $(1/c_2) * [d(p, next^{(c_2 s)}(p))]$.

Figure 5: Procedure 2: Estimating $(\ln n)/n$

1. $t_p \leftarrow$ estimate of $\ln n$, via Procedure 1;
2. $d_p \leftarrow$ estimate of $(\ln n)/n$ via Procedure 2;
3. $dmin_p \leftarrow c_3 * d_p$
4. Return $t_p \leftarrow c_4 * t_p$ and $dmin_p \leftarrow c_4 * dmin_p$

Figure 6: FindParametersI

The proof of Lemma 2 uses several lemmas concerning the *base hash function* h'; h' is the hash function which maps peers to points on the unit circle based on their IDs (i.e. IP addresses). As mentioned previously, we make the random oracle assumption [7] for the base hash function of the DHT. The first lemma is shown by Mahlki et al. [15].

Lemma 3. *With probability at least* $1 - 1/n$*: (property 1)* h' *has the property that for any peer,* p*,*

$$\ln n - \ln \ln n - 2 \leq \ln \left(\frac{1}{d(p, next(p))} \right) \leq 3 \ln n$$

Consider some interval I of the unit circle. We say that I is *anchored* if I has a peer point, p, at its counterclockwise endpoint. We say that p is the *anchor point* for I. The following lemma bounds the number of peers in anchored intervals of a certain size.

Lemma 4. *Let* $\alpha_1, \alpha_2, \epsilon$ *be fixed positive constants with* $\alpha_1 < \alpha_2$ *and* $0 \leq \epsilon \leq 1/2$*. Let* $C \geq 16/(\alpha_1 \epsilon^2)$*. Then for* n *sufficiently large, with probability at least* $1 - 1/n$*, the following (property 2) is true for* h'*:*

- *For any anchored interval* I *on the unit circle, if the number of peers that* I *contains other than the anchor point is greater than* $C\alpha_1 \ln n$ *and less than* $C\alpha_2 \ln n$*, then* I *is of length between* $C(1-\epsilon)\alpha_1(\ln n/n)$ *and* $C(1+\epsilon)\alpha_2(\ln n/n)$

1. $t_p \leftarrow$ estimate of $\ln n$, via Procedure 1;
2. $d_p \leftarrow$ estimate of $(\ln n)/n$ via Procedure 2;
3. $tmax_p \leftarrow c_3 * t_p$
4. Return $d_p \leftarrow c_4 * d_p$ and $tmax_p \leftarrow c_4 * tmax_p$

Figure 7: FindParametersII

Proof. We must show two facts are true with high probability. First, that no anchored interval I of length less than $C(1-\epsilon)\alpha_1(\ln n/n)$ contains greater than $C\alpha_1 \ln n$ peers other than the anchor point. Second, that no anchored interval I of length greater than $C(1+\epsilon)\alpha_2(\ln n/n)$ contains less than $C\alpha_2 \ln n$ peers other than the anchor point.

We start with the first fact. Consider some anchored interval I of length $C(1-\epsilon)\alpha_1(\ln n/n)$. Let X be a random variable giving the number of peer points other than the anchor which fall in I. Note that $E(X) = ((n-1)/n)C(1-\epsilon)\alpha_1(\ln n)$. Further, by Chernoff bounds, we know that for any $0 < \delta \leq 1$,

$$Pr(X \geq (1+\delta)E(X)) \leq e^{-\delta^2 E(X)/3}.$$

Setting $\delta = \epsilon$ implies that:

$$\begin{aligned} Pr(X \geq C\alpha_1 \ln n) &\leq e^{-\epsilon^2 E(X)/3} \\ &\leq e^{-\epsilon^2 C(1-\epsilon)\alpha_1/6} \\ &\leq 1/n^3. \end{aligned}$$

The second line in the above follows provided that $n \geq 2$. The last line follows since $C \geq 12/(\epsilon^2\alpha_1)$ and $1-\epsilon \geq 1/2$ (since $\epsilon \leq 1/2$)

There are exactly n anchored intervals of length $C(1-\epsilon)\alpha_1(\ln n/n)$. Thus a simple union bound shows that with probability no more than $1/n^2$, no anchored interval of length less than $C(1-\epsilon)\alpha_1(\ln n/n)$ contains greater than $C\alpha_1 \ln n$ peers.

Now we show the second fact is true with high probability. Consider some anchored interval I of length $C(1+\epsilon)\alpha_2(\ln n/n)$. Let X be a random variable giving the number of peer points other than the anchor which fall in I. Note that $E(X) = ((n-1)/n)C(1+\epsilon)\alpha_2(\ln n)$. Further, by Chernoff bounds, we know that for any $0 < \delta < 1$,

$$Pr(X \leq (1-\delta)E(X)) \leq e^{-\delta^2 E(X)/2}.$$

We want to choose a δ such that $(1-\delta)E(X) \geq C\alpha_2 \ln n$. Choosing $\delta = \epsilon/2$ ensures that this is true for n sufficiently large (specifically $n \geq \frac{2(1+\epsilon)}{\epsilon}$). Using this value for δ, we get that:

$$\begin{aligned} Pr(X \leq C\alpha_2 \ln n) &\leq e^{-\epsilon^2 E(X)/8} \\ &\leq \epsilon^2 C\alpha_2(\ln n)/16 \\ &\leq 1/n^3. \end{aligned}$$

The last line in the above follows since $C \geq 16/(\epsilon^2\alpha_2)$.

There are exactly n anchored intervals of length $C(1+\epsilon)\alpha_2(\ln n/n)$. Thus a simple union bound shows that with probability no more than $1/n^2$, no anchored interval of length greater than $C(1+\epsilon)\alpha_2(\ln n/n)$ contains less than $C\alpha_2 \ln n$ peers.

We have show that fact (1) fails to be true with probability $1/n^2$ and fact (2) fails to be true with probability $1/n^2$. Finally, a union bound gives that the probability that either fact is not true is no more than $2/n^2$. This probability is no more than $1/n$ provided that $n \geq 2$.

□

The following lemma bounds the size of any interval containing more than a certain number of peer points.

Lemma 5. *With probability greater than $1 - 1/n$, (property 3) h' has the property that any interval containing at least $8 \ln n$ peer points has length greater than $(\ln n)/n$.*

Proof. We will show that no interval of length $(\ln n)/n$ contains greater than or equal to $8 \ln n$ peer points. The analysis follows from the balls and bins paradigm. Partition the unit circle into disjoint consecutive intervals (bins) of length $(\ln n)/n$. Let X be the number of balls in any one bin. Then $E[X] = \ln n$. By the Chernoff bound, $Pr(X \geq (1+\delta)E[X]) < e^{-2E[X]} = 1/n^2$ for $\delta \geq 3$.

Let $\delta = 3$. With probability $1/n^2$, no consecutive pair of bins contains more than $2(1+\delta)E[X] = 8 \ln n$ peer points. A simple union bound then implies that no interval of length $(\ln n)/n$ in the unit circle contains greater than or equal to $8 \ln n$ peer points. □

We can now prove Lemma 2

Proof. Let h' be a random hash function mapping the n peers uniformly at random to the unit circle. Then by a simple union bound and Lemmas 3, 4, and 5, we know that with probability at least $1 - 3/n$, properties (1)–(3) hold. In the remainder of this proof, we will let p be an arbitrary peer and assume the three properties hold.

Let $c_1 = 1$ and $c_4 = 16$, then, by property (1), t_p will be of size at least $8 \ln n$. By property (2), if we set c_2 sufficiently large, we can ensure that the value returned by Procedure 2 will be no more than $4(\ln n/n)$. Then setting c_3 to be $1/64$ ensures that $dmin_p$ will be no larger than $(\ln n)/n$. We know that by property (3), any interval containing at least $8 \ln n$ peer points has length greater than $(\ln n)/n$. Thus for every peer p, every interval containing t_p peers has length at least $dmin_p$. Finally, we note that the constants have been set in such a way that $t_p = \Theta(\ln n)$ and $dmin_p = \Theta((\ln n)/n)$. □

3.2 Analysis of FindParametersII

In this section, we will prove the following lemma about the procedure FindParametersII.

Lemma 6. *Assume n peers are distributed uniformly at random on the unit circle of a DHT. Then there exist settings for the constants c_1, c_2, c_3, c_4 in the procedure FindParametersII which ensure the following with probability at least $1 - 3/n$. For every peer p, d_p and $tmax_p$ are chosen such that:*

- *No interval of length d_p contains more than $tmax_p$ peers*
- *$tmax_p = \Theta(\ln n)$ and $d_p = \Theta((\ln n)/n)$.*

We will make use of the following simple corollary which follows directly from Lemma 5.

Corollary 7. *Property (3) implies that no interval of length $(\ln n)/n$ contains greater than or equal to $8 \ln n$ peer points.*

We now present the proof of Lemma 6.

Proof. Let h' be a random hash function mapping the n peers uniformly at random to the unit circle. Then by a simple union bound and Lemmas 3, 4, and 5, we know that with probability at least $1 - 3/n$, properties (1)–(3) hold. In the remainder of this proof, we will let p be an arbitrary peer and assume the three properties hold.

Let $c_1 = 1$, then by property (2), if we set c_2 sufficiently large, we can ensure that the value returned by Procedure 2 will be less than or equal to $4(\ln n)/n$. If we then set $c_4 = 1/4$, we can ensure that $d_p \leq (\ln n)/n$. Now if we set $c_3 = 64$, by Property (1), we can ensure that $tmax_p \geq 8 \ln n$. We know that by property (3) and Corollary 7, no interval of length $(\ln n)/n$ contains greater than or equal to $8 \ln n$ peer points. Thus for every peer p, no interval of length d_p contains more than $tmax_p$ peer points. Finally, we note that the constants have been set in such a way that $d_p = \Theta((\ln n)/n)$ and $tmax_p = \Theta(\ln n)$. □

3.3 Analysis of Algorithm *Peer Count*

In this section, we prove the following lemma.

Lemma 8. *Assume in an execution of algorithm* Peer Count *that t_p and $dmin_p$ are chosen so that every interval containing t_p peers has length at least $dmin_p$. Then algorithm* Peer Count *has the following properties.*

- *Each peer is chosen with the same probability, namely $dmin_p/t_p$.*
- *The expected number of rounds is $t_p/(n \cdot dmin_p)$.*

- *For any positive integer r, the probability that the number of rounds is greater than r is $(1 - n \cdot dmin_p/t_p)^r$*
- *There is exactly one call to h per round. The number of calls to next per round is t_p except for the last round, where it may be less than t_p.*

We will prove this lemma as follows. We will say that *Peer Count* assigns a point x on the unit circle to a peer q if *Peer Count* returns q when x is the random number chosen in step 1. In the proofs below, we will fix the peer p that is running the peer count algorithm and will let λ be $dmin_p/t_p$ as in the second step of *Peer Count*. We will then show that *Peer Count* assigns to each peer a set of disjoint intervals whose lengths sum to λ.

Lemma 9. *For any point r on the unit circle, if r is assigned by* Peer Count *to a peer q, then $d(r,q) < \lambda num(r,q)$ and $num(r,q) \leq t_p$. If there is more than one such peer, then the algorithm assigns r to the closest one, i.e., the one such that $d(r,q)$ is minimal.*

Proof. In the loop at line 6, the algorithm visits a succession of peer points going clockwise from r. Let q_i represent the peer whose peer point is the i^{th} encountered (here, $q_1 = h(r)$). In line 5, T is set to $d(r,q_1) - \lambda$. It is easy to see by induction that at the i^{th} repetition of line 7, $T = d(r,q_{i+1}) - \lambda(i+1) = d(r,q_{i+1}) - \lambda num(r,q_{i+1})$. The algorithm returns the first peer q_i such that $T < 0$, i.e., $d(r,q_i) < \lambda num(r,q_i)$, provided that such a peer is encountered within t_p peer points of r.

□

For any peer q, let $Int(q) = (x,q]$ be the half-closed interval on the unit circle whose endpoint x is the closest point counterclockwise from q such that $d(x,q) \geq \lambda num(x,q)$.

Lemma 10. *Let q, q' be any peers such that $num(Int(q)) \leq t_p$ and $q \neq q'$. Then:*

1. *Every point assigned by the algorithm to q lies in $Int(q)$.*
2. *Every point in $Int(q)$ is assigned by the algorithm to a peer whose peer point lies in $Int(q)$.*
3. *Either $Int(q) \subset Int(q')$, $Int(q') \subset Int(q)$, or $Int(q) \cap Int(q') = \emptyset$.*

Proof. Proof of (1): Let $Int(q) = (x,q]$. Let y be a point assigned to q. Then $d(y,q) < \lambda num(y,q)$, by Lemma 9. Assume to the contrary that y lies outside $Int(q)$.

We first look at the case that there is no peer point in $[y,x]$. Then $d(y,q) = d(y,x) + d(x,q) \geq d(x,q) \geq \lambda num(x,q) = \lambda num(y,q)$, contradicting the assumption that the algorithm would have assigned y to q.

Alternatively, let q' be the peer whose peer point is closest to x in $[y,x]$ (or equal to x if x is a peer point). By assumption, since y was assigned to

q, $d(y,q) < \lambda num(y,q)$. Now, $d(y,q) = d(y,q') + d(q',q)$ and $num(y,q) = num(y,q') + num(q',q)$. Hence we have

$$d(y,q') + d(q',q) < \lambda num(y,q') + \lambda num(q',q).$$

Since $d(q',q) \geq d(x,q) \geq \lambda num(x,q)$ and $num(q',q) = num(x,q)$, the above inequality is preserved when we subtract $d(q',q)$ from the left-hand side and $\lambda num(q',q)$ from the right-hand side. This implies:

$$d(y,q') < \lambda num(y,q').$$

By Lemma 9, the algorithm would have assigned y to q' since $d(y,q') < d(y,q)$, contradicting our assumption.

Proof of (2): This follows from the fact that every point y in $lnt(q)$ has the property that $d(y,q) < \lambda num(y,q)$. Hence by Lemma 9, y is either assigned to q or some closer peer in $[y,q]$.

Proof of (3): This is similar in technique to the proof of (1) and is left to the reader.

□

Lemma 11. *The set of intervals assigned to any peer q with $num(Int(q)) \leq t_p$ has total length λ.*

Proof. The proof is by induction on the size of $num(Int(q))$ where q is any peer.

Base Case: $num(Int(q)) = 1$. In this case, $Int(q) = (q - \lambda, q]$. Lemma 10 (2) implies that every point in $Int(q)$ is assigned to q and Lemma 10 (1) implies that no other point is assigned to q so the single interval assigned to q has length λ.

Induction step: Suppose $num(Int(q)) = k$. Then there are $k-1$ peer points within $Int(q)$ excluding q. By Lemma 10(3), each of these peer points q' have $Int(q') \subset Int(q)$. Since $Int(q')$ does not contain q, $num(Int(q')) < k$. By the induction assumption, each peer q' is assigned an interval of length λ, for a total of $(k-1)\lambda$. By Lemma 10 (2), every point in $Int(q)$ is assigned to a peer in $Int(q)$. Hence since $d(Int(q)) = \lambda k$, $k\lambda - (k-1)\lambda = \lambda$ has been assigned to q. By lemma 10 (1), no other points on the unit circle have been assigned to q. Hence q has been assigned a set of intervals whose lengths add up to λ. □

We now give the proof of lemma 8.

Proof. By Lemma 11, we need only show that for any peer q, $num(Int(q)) \leq t_p$. Assume to the contrary that for some peer q, $num(Int(q)) > t_p$. Let $Int(q) = (x,q]$ for some point x. Let y be the closest point to q in $Int(q)$ such that $num(y,q) = t_p$. Note that y is closer to q than x. By our assumption, $d(y,q) \geq dmin_p$. But we know that $dmin_p = \lambda t_p$. Thus $d(y,q) \geq \lambda t_p = \lambda num(y,q)$. This contradicts the fact that x is the closest point counterclockwise to q such

that $d(x,q) \geq \lambda num(x,q)$. Thus, each peer is chosen with probability exactly λ.

By the above argument, the probability that some peer is chosen in a given round is $n\lambda$, and so the expected number of rounds is $1/(n\lambda) = t_p/ndmin_p$. Further the probability that the number of rounds is greater than r is $(1-n\lambda)^r$. The number of calls to *next* is easily seen to be t_p in all but possibly the last round, which may be completed without all t_p calls. □

3.4 Analysis of Algorithm *Arc Length*

Lemma 12. *Assume in an execution of algorithm* Arc Length *that $tmax_p$ and d_p are chosen so that every interval of length d_p contains no more than $tmax_p$ peers. Then algorithm* Arc Length *has the following properties.*

- *It chooses each peer with equal probability, namely $d_p/tmax_p$*
- *The expected number of rounds is $tmax_p/(nd_p)$.*
- *For any positive interger r, the probability that the number of rounds is greater than r is $(1 - nd_p/tmax_p)^r$*
- *The number of calls to h per round is 1 and the number of calls to next in each round is no more than $tmax_p$.*

Proof. We first show that any peer q is selected in a given round with probability $d_p/tmax_p$, hence each is selected with equal probability. Let ξ_1 be the event that q is within distance d_p of r. Then $Pr[\xi_1] = d_p$. Let ξ_2 be the event that q is the peer returned by the algorithm. Then $Pr[\xi_2|\xi_1] = 1/tmax_p$ (since any interval of length d_p contains no more than $tmax_p$ peers). Since $Pr[\xi_2] = Pr[\xi_2|\xi_1]Pr[\xi_1]$, we have that $Pr(\xi_2) = d_p/tmax_p$.

The probability that any peer is selected in a given round is thus $nd_p/tmax_p$. This means that the expected number of rounds is $tmax_p/(nd_p)$ and that the probability that the number of rounds is greater than r is $(1 - nd_p/tmax_p)^r$ for any positive integer r. The number of calls to *next* per round is never more than $tmax_p$. □

3.5 Proof of Theorem 1

We now give the proof of Theorem 1. We first show the theorem holds for algorithm *Peer Count*. To see this, note that by Lemma 2, with probability at least $1-3/n$, $t_p = \theta(\ln n)$ and $dmin_p = \theta((\ln n)/n)$ and every interval containing t_p peers has length at least $dmin_p$. Thus, Lemma 8 implies that algorithm *Peer Count* is correct and that, in expectation, it has latency $\ln n$ and sends $\ln n$ messages. Further, if in Lemma 8 we set $r = \theta(\ln n)$, it implies that with high probability, the number of rounds is $\theta(\ln n)$. Thus, with high probability, the latency and number of messages sent are both $O(ln^2 n)$.

We next show that Theorem 1 holds for algorithm *Arc Length*. First, we note that by Lemma 6, with probability at least $1 - 3/n$, $tmax_p = \Theta(\ln n)$ and

	Peer Count		*Arc Length*	
	Range Tested	Step	Range Tested	Step
c_1	$2-5$	1	$2-5$	1
c_2	$2-5$	1	$2-5$	1
c_3	0.1 - 0.5	0.1	$2-5$	1
c_4	$2-5$	1	$2-5$	1

Figure 8: Range of Values Table

	Peer Count	*Arc Length*
c_1	5.0	2.0
c_2	5.0	4.0
c_3	0.2	4.0
c_4	4.0	2.0

Figure 9: Selected Values Table

$d_p = \Theta((\ln n)/n)$ and that no interval of length d_p contains more than $tmax_p$ peers. Thus, Lemma 12, implies that algorithm *Arc Length* is correct and that, in expectation, it has latency $\ln n$ and sends $\ln n$ messages. Further, if in Lemma 12 we set $r = \theta(\ln n)$, it implies that with high probability, the number of rounds is $\theta(\ln n)$. Thus, with high probability, the latency and number of messages sent are both $O(ln^2 n)$.

4 Empirical Results

The failure probability of our algorithms depends critically on the values of the constants c_1, c_2, c_3, c_4. The relationship between these constants, the failure probability and the latency is given by a non-linear system of equations. Since finding an *optimal* solution is computationally intractable, our goal is to find settings for the constants which ensure that the algorithms are both 1) correct for a large number of randomly generated DHTs and 2) have low latency.

4.1 Maintaining Correctness

For *Peer Count*, a set of constants is correct for a DHT if, for all peers p, no interval containing t_p consecutive peers has length less than $dmin_p$. For *Arc Length*, a set of constants is correct for a DHT if, for all peers p, no interval of length d_p contains more than $tmax_p$ peers. If these conditions hold, then any peer in the DHT which executes the algorithms will select a peer uniformly at random.

	Peer Count		*Arc Length*	
	Range Tested	Step	Peer Count	Step
c_1	6.0-13.0	1.0	3.0-10.0	1.0
c_2	6.0-13.0	1.0	5.0-12.0	1.0
c_3	0.04-0.18	0.02	5.0-12.0	1.0
c_4	5.0-12.0	1.0	3.0-10.0	1.0

Figure 10: Range of values used for investigating latency.

4.2 Setting the Constants

To find candidate constant settings, we explored discrete points in a large space of possible constant values. The range of values table in figure 8 shows the range of values tested for each algorithm. We only kept those settings which we verified to be correct for $1,000$ random DHTs containing $10,000$ peers. In other words, for all 10 million peers in the $1,000$ DHTs, we verified that *Peer Count* and *Arc Length* would run correctly on each peer. From this we concluded that the error probability of the algorithms, using those settings, was small. Note that the probability of error decreases as n increases.

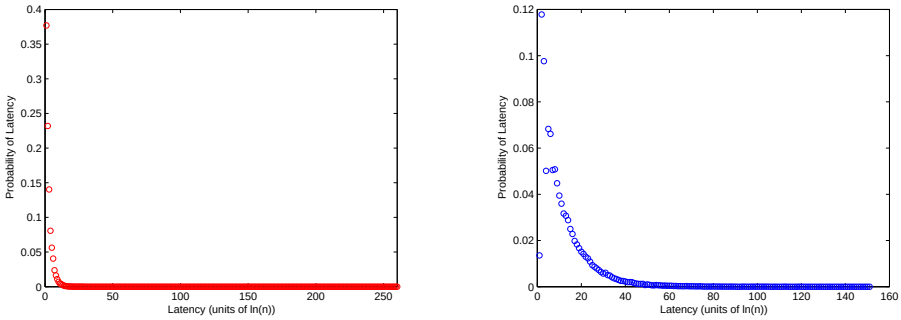
4.3 Measuring Latency

From all constant settings which passed the empirical test described in the previous section, we chose one for each algorithm which minimized the average latency over many trials. The selected values table in Figure 9 gives the settings chosen for the algorithms *Peer Count* and *Arc Length*, respectively.

In our empirical tests, to get specific numerical values for latency and message costs for calls to *next* and h, we assume Chord is the underlying DHT. Therefore, the latency for a call to either of our algorithms is defined as ([# of rounds]$\cdot \log n$)+[# of calls to next] excluding the calls to *next* incurred by $FindParametersI$ and $FindParametersII$. We expect that a peer will need to update its parameters infrequently and, therefore, estimate the latency cost based only on calls to *next* and h within the main 'while' loop of each algorithm.

4.4 Computational Results and Analysis

In order to find the mean latency for both *Peer Count* and *Arc Length*, 100 random DHTs were generated each with $10,000$ peers. For each DHT, $10,000$ executions of each algorithm were performed by peers chosen uniformly at random. The mean latency for *Peer Count* was found to be $20.02 \log n$ and the mean latency for *Arc Length* was found to be $10.01 \log n$. The latency distribution for both *Peer Count* and *Arc Length* is presented in Figure 11. Both distributions are tightly concentrated around their means.



(a) **(b)**

Figure 11: (a-b) The latency distribution for *Peer Count* and *Arc Length*, respectively.

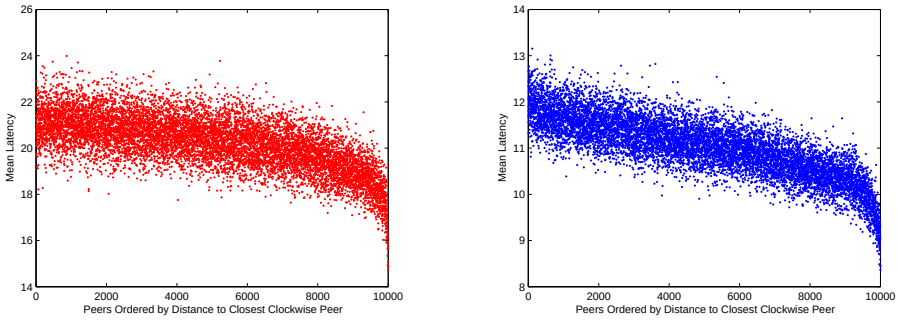
4.4.1 Latency and Clockwise Arc Length

We investigate the relationship between the latency of a run of the algorithms *Peer Count* and *Arc Length* and the length of the arc between the peer running the algorithm and the closest clockwise peer point (we will call this the clockwise arc length of the peer). We generated 100 random DHTs each with $10,000$ peers. For each DHT, the average latency incurred by each peer over a total of 10 executions of *Peer Count* and *Arc Length* was recorded. The peers were then ordered by clockwise arc length. The results over all 100 DHTs were averaged and plotted in Figure 12 (a-b). As evidenced by these plots, there is a decrease in latency as the clockwise arc length increases. This is not surprising given that as the clockwise arc length of a peer increases, that peer will provide tighter parameter values to the algorithms *Peer Count* and *Arc Length* .

4.4.2 Latency, Constants and Clockwise Arc Length

In this section, we examine the effect of the constants c_1, c_2, c_3, c_4 on the latency of peers with differing clockwise arc lengths. For instance, one may question whether some optimal setting of the constants for peers with relatively small clockwise arc length differs from some optimal setting for peers with larger clockwise arc length. If so, peers might benefit from individually setting their constants according to some function of their clockwise arc length.

Figure 13 (a-d) and Figure 14 (a-d) depict the latency as measured against clockwise arc length and constants c_1, c_2, c_3, c_4 for *Peer Count* and *Arc Length*, respectively. For each constant, 10 random DHTs were created each with $10,000$ peers. Each peer of a random DHT executed *Peer Count* or *Arc Length* 5 times, respectively. This provided data on average latency versus forward arc legnth for a certain constant value and a best fit line was obtained. Over the constant values tested, these best fit lines define the interpolated surfaces observed in



(a) **(b)**

Figure 12: (a-b) The mean latency values versus the peers ordered by increasing distance to the closest clockwise peer for *Peer Count* and *Arc Length*, respectively.

Figure 13 (a-d) and 14 (a-d).

The table in Figure 10 provides the different values for c_1, c_2, c_3, c_4 that were tested for *Peer Count* and *Arc Length*. The resulting plots in Figures 13 and 14 allow for a visual inspection of effects due to changing the value of any single constant setting. Both *Peer Count* and *Arc Length* fail to exhibit any significant change in latency over the values for clockwise arc length and constants c_1 and c_2. Increasing these constants can help peers obtain more accurate values of $\ln n$ and $\ln n/n$. However, this does not significantly improve the overall latency of the algorithms.

As the constants c_3 and c_4 are varied, we see a marked change in latency, which is expected. However, there is no significant change in slope in the plane defined by the latency and clockwise arc length axes for any of these plots. These results suggest that, within the range of constants chosen, there are no settings which improve latency costs regardless of clockwise arc length.

4.4.3 Summary

Overall, the results of our empirical tests match our theoretical predictions. For *Peer Count*, the average number of rounds is 4.85, with an average latency of $20.02 \log n$. For *Arc Length*, the average number of rounds is about 3.90, with an average latency of $10.01 \log n$. The results suggest that algorithm *Arc Length* requires significantly less bandwidth and latency than algorithm *Peer Count*. Both algorithms exhibit a tendency to favor peers with larger clockwise arc length. Experimental evidence suggests that this behavior occurs over a range of settings for the constants. The simplicity and efficiency of *Arc Length* makes it an attractive choice over *Peer Count*. One possible downside of *Arc Length* in comparison with *Peer Count* is that it requires more random bits. In particular, *Arc Length* requires $\log \log n$ more random bits per round than *Peer Count*.

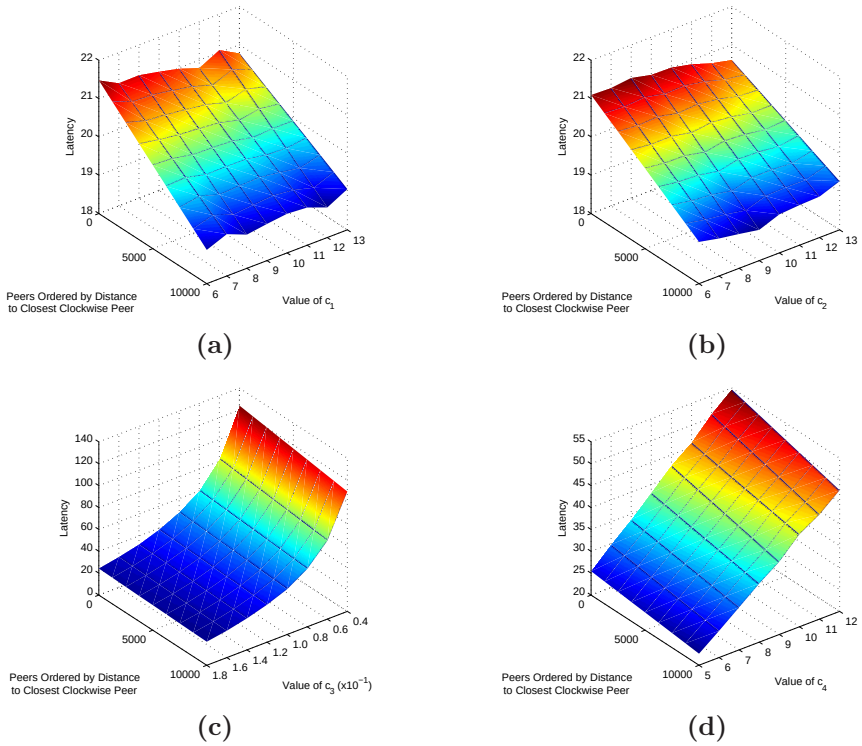


Figure 13: Plots of latency values measured against peers ordered by increasing distance to the closest clockwise peer and varying constants for *Peer Count*.

5 An Application to Unstructured Networks

We note that algorithm *Arc Length* can be modified to choose a random peer in an unstructured network as follows. Assume that all peers have IDs chosen independently and uniformly at random between 0 and 1 and that the peer p running the algorithm has estimates d_p and $tmax_p$ as described in Section 2. The peer p picks a random number r between 0 and 1. It broadcasts r and the number d_p to all peers in the network. All peers which have IDs within distance d_p of r then respond to this broadcast. Peer p chooses a random number between 1 and $tmax_p$ and then chooses the x-th closest peer to r among all peers that responded to its broadcast, if at least x peers responded. Otherwise, it repeats the algorithm. While p must broadcast to the entire network, we expect only $\Theta(\log n)$ peers to have to respond to p's broadcast. This modified algorithm might be of particular interest in a sensor or mobile network application where p is a powered node and the other nodes are unpowered nodes. In this case, we are able to minimize the number of unpowered nodes that have to send messages

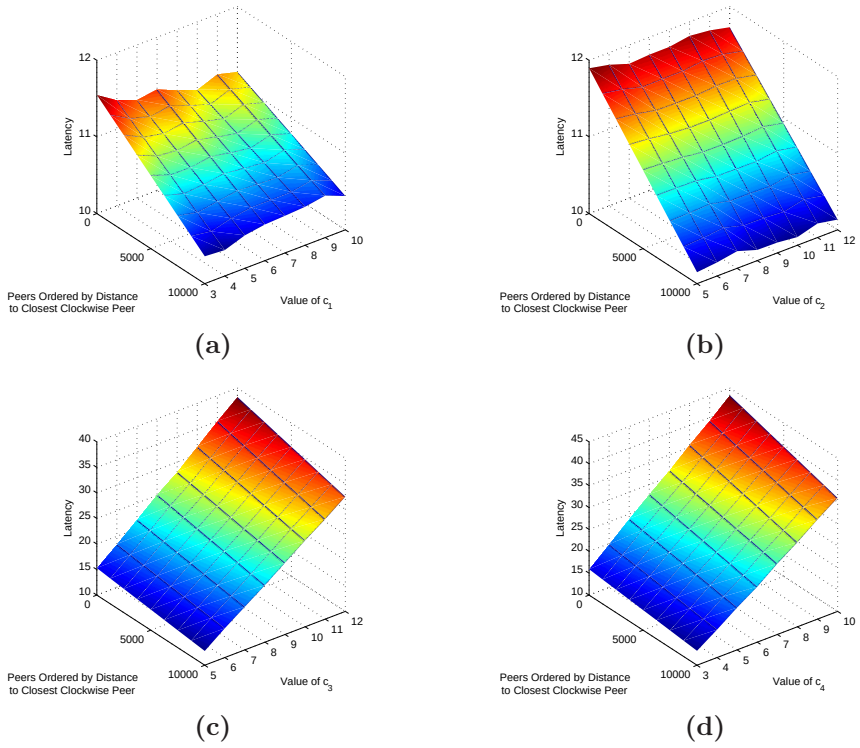


Figure 14: Plots of latency values measured against peers ordered by increasing clockwise arc length and varying constants for *Arc Length*.

to p and so are able to conserve battery power.

6 Conclusion and Future Work

We have presented the first algorithms for choosing a peer uniformly at random from the set of all peers in a DHT. We have shown that these algorithms have expected latency and message cost which is $O(\log n)$. We have also shown that, in practice, the algorithms are quite efficient. Several open problems remain including the following:

- Many peer-to-peer networks like Gnutella have much less structure than a DHT. Based on empirical studies [23], it seems reasonable to make the assumption that these semi-structured networks at least have good expansion properties. Can we design efficient algorithms for choosing a random peer in such semi-structured peer-to-peer networks with good expansion properties? In particular, can we do better than a random walk in the

sense that we guarantee that the peer selected is selected precisely uniformly at random? Another interesting question is: Can we design efficient algorithms by assuming a formal model of network creation and maintenance such as the model in [4].

- Can we design an algorithm to efficiently choose a node uniformly at random in a sensor network? In sensor networks, connections are determined by a distance metric and the points are typically randomly distributed. In such networks, power consumption is also another critical resource to conserve.

References

[1] Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A Scalable Content-Addressable Network. Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (2001)

[2] Rowstron, A., Druschel, P.: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, (2001) 329–350

[3] Zhao, B., Kubiatowicz, J., Joseph, A.: Tapestry: An Infrastructure for Fault-Resilient Wide-Area Location and Routing. University of California at Berkeley Technical Report, UCB//CSD-01-1141, (April 2001)

[4] Gopal Pandurangan and Prabhakar Raghavan and Eli Upfal. Building Low-diameter P2P Networks. In *Symposium on Theory of Computation (STOC) 2001.*

[5] Amos Fiat, Jared Saia and Maxwell Young. Making Chord Robust to Byzantine Attack, In *Proceedings of the European Symposium on Algorithms(ESA), 2005.*

[6] Micah Alder, Eran Halperin, Richard Karp, and Vijay Vazirani. A stochastic process on the hypercube with applications to peer-to-peer networks. In *ACM Symposium on Theory of Computing (STOC)*, 2003.

[7] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *The First ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[8] John Byers, Jeffrey Considine, and Michael Mitzenmacher. Simple load balancing for distributed hash tables. In *Proceedings of the Second Internation Peer to Peer Symposium (IPTPS)*, 2003.

[9] Christos Gkantsidis, Milena Mihail, and Amin Saberi. Random walks in peer-to-peer networks. In *Conference of the IEEE Communications Society (INFOCOM)*, 2004.

[10] David Karger and Matthias Ruhl. New algorithms for load balancing in peer-to-peer systems. In *Proceedings of the Fourth Internation Peer to Peer Symposium (IPTPS)*, 2004.

[11] David Karger and Matthias Ruhl. Finding nearest neighbors in growth-restricted metrics. In *ACM Symposium on Theory of Computing (STOC)*, 2002.

[12] Valerie King and Jared Saia. Choosing a random peer. In *ACM Conference on the Principles of Distributed Computing(PODC)*, 2004.

[13] C. Law and K.-Y. Siu. Distributed construction of random expander graphs. In *Conference of the IEEE Communications Society (INFOCOM)*, 2003.

[14] Scott Lewis and Jared Saia. Scalable byzantine agreement. Technical report, University of New Mexico, 2004.

[15] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic lookup network. In *ACM Conference on the Principles of Distributed Computing(PODC)*, 2002.

[16] Gurmeet Singh Manku. Routing networks for distributed hash tables. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 133–142. ACM Press, 2003.

[17] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[18] Rajeev Motwani and Prbhakar Raghavan. *Randomized Algorithms*, chapter 5.3. Cambridge University Press, 1995.

[19] Rajeev Motwani and Prbhakar Raghavan. *Randomized Algorithms*, chapter 4. Cambridge University Press, 1995.

[20] Moni Naor and Udi Weider. Novel architectures for p2p applications: the continuous-discrete approach. In *Symposium on Parallel Algorithms and Architecture (SPAA)*, 2003.

[21] 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*.

[22] Stefan Saroiu, Krishna P. Gummadi, Richard J. Dunn, Steven D. Gribble, and Henry M. Levy. An analysis of internet content delivery systems. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.

[23] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of Multimedia Computing and Networking*, 2002.

[24] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of the ACM SIGCOMM 2001 Technical Conference*, San Diego, CA, USA, August 2001.