On Algorithms for Choosing a Random Peer

Valerie King ∗ Scott Lewis † Jared Saia †

Abstract
In this paper, we study the problem of choosing a peer uniformly at random from the set of all peers in a distributed hash table (DHT). We present two new algorithms to solve this problem and show that these algorithms have good theoretical and empirical properties.

1 Introduction
In this paper, we address the problem of choosing a peer uniformly at random from the set of all peers in a Distributed Hash Table (DHT). 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:

• Data Collection: 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 [? , ?]; and environmental data, e.g., for sensor networks.

• Supporting Randomized Algorithms: We know of two randomized algorithms for peer-to-peer systems which require a function for choosing a random peer. The first algorithm ensures good load-balancing of computational tasks across the peers in a network [?]. The second algorithm provides a scalable solution to the Byzantine agreement problem [?]. Both algorithms critically rely on the existence of a function for choosing a random peer, but unfortunately, both results only suggest heuristics to approximate such a function.

• Creating Random Links: Consider a network where every node has a small number of links to other random nodes. Such a network 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 [?]. A function for choosing a random peer allows for simple creation and maintenance of random links, and these random links provide an extra measure of robustness.

We use the following DHT model [?]. We assume that the “key space” of the DHT is scaled so it is in the range (0,1]. We will think of the DHT as a circle with unit circumference, which we will call the unit circle. We assume that $n$ peers are connected in the DHT and that all of the $n$ peers are mapped to locations on the unit circle which we call peer points. We assume that the $n$ peer points are distributed uniformly at random on the unit circle $\mathbb{S}^1$. The DHT provides two basic operations: $h$ and $\text{next}$. 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 $p$, $\text{next}(p)$ returns the peer whose peer point is closest in clockwise distance to $p$’s peer point. We assume that single applications of $\text{next}$ and of $h$ have latencies of 1 and $\log n$, respectively, and require 1 and $\log n$, respectively, messages to be sent.

Our problem then is to design a scalable, distributed function which chooses a peer uniformly at random from the set of all peers in the DHT. We want this function to use only the basic DHT operations $h$ and $\text{next}$ and we want it to be scalable in the sense that latency 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 $h(x)$. Unfortunately, this heuristic is biased. The probability that a peer $p$ is chosen is proportional to the length of the arc between the peer point for $p$ and the closest counter-clockwise peer point. The lengths

As is standard, we use the random oracle model [?] for the base hash function of the DHT.

Throughout the paper, we will use log to represent log base 2.
of these arcs vary widely. With high probability, the longest arc is of length $\Theta(\log n/n)$ [?] and the shortest arc is of length $\Theta(1/n^2)$ [?]. Thus, the peer with the longest arc will be chosen $\Theta(n\log n)$ times more frequently than the peer with the shortest arc. To remove this bias, we require a more sophisticated algorithm.

1.1 Our Results

Our paper builds on a result from [?]. In that paper, an algorithm, which we will call Peer Count, is presented for choosing a random peer in a DHT. For any base hash function of the DHT, with probability $1 - 3/n$, Peer Count always chooses peers uniformly at random every time it is called by any peer in the DHT. Moreover, the algorithm has expected latency and message cost which is $O(\log n)$. Unfortunately the hidden constants in these asymptotic terms are somewhat large.

The contributions of this paper are threefold:

- We give a new version of Peer Count which tightens parameters to reduce hidden constants.
- We introduce Arc Length, a new algorithm for selecting a random peer which provably has the same properties and asymptotic performance as Peer Count and which performs slightly better in practice.
- We empirically test both Peer Count and Arc Length 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 Peer Count is $19.1 \log n$ and the average latency and message cost of a single call to Arc Length is $11.4 \log n$. These new algorithms both give over an order of magnitude improvement over the old version of Peer Count in [?].

1.2 Related Work

Gkansidis et. al. address the problem of choosing a random peer in a peer-to-peer system [?]. 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 [?] 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 [?] for a technique involving virtual nodes in which each peer maps to $O(\log n)$ peer points on the unit circle and [?, ?, ?, ?] for other techniques. Generally these techniques work by dynamically “re-assigning” hash space among the peers to ensure that no peer is ever responsible for too large a portion.

We have assumed a standard DHT which has no load-balancing extensions. We make this assumption for two reasons. First, we would like our protocol 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 DHT can be easily adapted to a DHT which has load-balancing extensions. Second, we want our results to hold even in the presence of malicious faults and we are not aware of any DHTs with load-balancing extensions which are provably robust to malicious faults.

1.3 Notation

For any two points $x$ and $y$ on the unit circle, we let $\text{dist}(x, y)$ be the distance from $x$ to $y$ traveling clockwise along the unit circle (i.e. if $y \geq x$, then $\text{dist}(x, y) = y - x$ else $\text{dist}(x, y) = (1 - x) + y$). Let $\text{num}(x, y)$ denote the number of peer points in the half-closed interval $(x, y]$ traveling clockwise from $x$ to $y$ along the unit circle.

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 $\text{next}$ returns the $k^{th}$ next peer in the clockwise ordering around the circle from $p$ and is denoted $\text{next}^{(k)}$.

The rest of this paper is laid out as follows. In Section 2 we give our two algorithms for choosing a random peer. In Section 3 we describe our empirical results for these two algorithms. We conclude and give directions for future work in Section 4. We give most of the proofs of correctness in Section A.

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 $\text{dmin}_p$ such that with high probability, no interval containing $t_p$ peers has length less than $\text{dmin}_p$. Arc Length depends on the ability of each peer $p$ to independently determine a length $d_p$ and a number
that no interval containing $t_p$ contains more than $t_{max_p}$ peers. In the next subsection, we show how these parameters can be chosen such that $t_p$ and $t_{max_p}$ are both $\Theta(\log n)$ and $d_{min_p}$ and $d_p$ are $\Theta(\log n/n)$.

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

### 2.1 Peer Count algorithm

This new version of Peer Count [?] tightens some of the parameters in the original algorithm in order to improve performance. In particular, $6\ln n'$ is replaced by $t_p$ and $\lambda$ is set equal to $d_{min_p}/t_p$. The algorithm is presented formally in Figure 1.

A peer $p$ initially calls $FindParametersI$ to determine values for $d_{min_p}$ and $t_p$ and sets $\lambda$ to $d_{min_p}/t_p$. 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 $dist(r,p') < \lambda num(r,p')$ 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.

**Lemma 1.** Assume in an execution of the Peer Count algorithm that $t_p$ and $d_{min_p}$ are chosen so that no interval containing $t_p$ peers has length less than $d_{min_p}$. Then the algorithm has the following properties: each peer is chosen with the same probability, namely $\lambda$; the expected number of rounds and number of calls to $h$ is $t_p/(\lambda d_{min_p})$; and the number of calls to $next$ per round is $t_p$ except for the last round, where it may be less than $t_p$.

**Proof.** The proof of correctness follows the one in paper [?] when $d_{min_p}$ and $t_p$ are substituted as explained above. It shows that each peer is chosen with probability $\lambda$, which implies that the probability that any peer is chosen in a given round is $n\lambda$, and the expected number of rounds is $1/(n\lambda) = t_p/d_{min_p}$. 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.

### 2.2 Arc Length algorithm

The Arc Length algorithm is presented formally in Figure 2 and we give an overview here.

A peer $p$ calls $FindParametersII$ to select parameters $d_p$ and $t_{max_p}$ such that with high probability, no interval of length $d_p$ contains more than $t_{max_p}$ peers. Then the algorithm enters a loop in which it selects a random number $r$ from $(0,1]$ and a random integer $x$ in $[1,t_{max_p}]$. 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) $dist(r,p') \leq d_p$, then $p'$ is returned; otherwise the loop is repeated. One execution of a loop is referred to as a round.

**Lemma 2.** If no interval of length $d_p$ contains more than $t_{max_p}$ peers, then the Arc Length algorithm chooses each peer with equal probability. The expected number of rounds and number of calls to $h$ is...
\(t \max_p/(nd_p)\). The expected number of calls to next in each round is \(nd_p\).

Proof. We first show that any peer \(p_i\) is selected in a given round with probability \(d_p/t \max_p\), hence each is selected with equal probability. Let \(\xi_1\) be the event that \(p_i\) is within distance \(d_p\) of \(r\). Then \(Pr[\xi_1] = d_p\). Let \(\xi_2\) be the event that \(p_i\) is the peer returned by the algorithm. Then \(Pr[\xi_2|\xi_1] = 1/t \max_p\). Since \(Pr[\xi_2] = Pr[\xi_2|\xi_1]Pr[\xi_1]\), we have that \(Pr(\xi_2) = d_p/t \max_p\).

The probability that any peer is selected in a given round is thus \(nd_p/t \max_p\) and the expected number of rounds and calls to \(h\) is thus \(t \max_p/(nd_p)\). The expected number of calls to next is the minimum of the expected number of peers in an interval of length \(d_p\) which is \(nd_p\) and the quantity \(t \max_p/2\).

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 \([?]\) and \([?]\). In Figure 3, 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 \([?,?]\). 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 4, Procedure 2 generalizes the algorithm from \([?,?]\) by introducing the constant \(C_2\).

1. \(p \leftarrow \text{id of self}\);
2. \(\hat{n} \leftarrow C_1/\text{dist}(p, \text{next}(C_1)(p))\);
3. Return \(\text{ln } \hat{n}\);

Figure 3: Procedure 1: Estimating \(\ln n\)

The procedures FindParametersI and FindParametersII are given in Figure 5 and Figure 6, respectively. These procedures first get estimates of \(\ln n\) and \(\ln n/n\) and then compute \(t_p\) and \(d \min_p\) (respectively, \(d_p\) and \(t \max_p\)).

2.4 Setting the Constants

We choose the values for the \(C_1, C_2, C_3, C_4\), so that: 1) the Peer Count and Arc Length algorithms are correct with high probability; and 2) the latency of Peer Count and Arc Length is small. For (1), we require that with probability \(1 - 1/n\),

- No interval containing \(t_p\) peers has length less than \(d \min_p\);
- No interval of length \(d_p\) contains more than \(t \max_p\) peers.

Theorem 11 shows that there is a setting of the constants \(C_1, C_2, C_3, C_4\) for Peer Count which ensures that, with high probability, no interval containing \(t_p\) peers has length less than \(d \min_p\). Theorem 13 shows that there is a setting of the constants \(C_1, C_2, C_3, C_4\) for Arc Length which ensures that, with high probability, no interval of length \(d_p\) contains more than \(t \max_p\) peers.

3 Empirical Tests

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 hard, our goal is to find settings for the constants which ensure that the algorithms are both 1) correct for a large number of random DHTs and 2) have low latency.
1. \( t_p \leftarrow \text{estimate of } \ln n, \text{via Procedure 1} \);
2. \( d_p \leftarrow \text{estimate of } (\ln n)/n \text{ via Procedure 2} \);
3. \( t_{max,p} \leftarrow C_3 * t_p \)
4. Return \( d_p \leftarrow C_4 * d_p \) and \( t_{max,p} \leftarrow C_4 * t_{max,p} \)

**Figure 6: FindParametersII**

<table>
<thead>
<tr>
<th>Peer Count</th>
<th>Arc Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>Step</td>
</tr>
<tr>
<td>( C_1 )</td>
<td>2 - 5</td>
</tr>
<tr>
<td>( C_2 )</td>
<td>2 - 5</td>
</tr>
<tr>
<td>( C_3 )</td>
<td>0.1 - 0.5</td>
</tr>
<tr>
<td>( C_4 )</td>
<td>2 - 5</td>
</tr>
</tbody>
</table>

**Figure 7: Range of Values for Constants**

### 3.1 Maintaining Correctness

For Peer Count, a set of constants, \( C_1, C_2, C_3, C_4 \), is correct for a DHT if, for all peers \( p \), no interval containing \( t_p \) consecutive peers has length less than \( d_{min,p} \). 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 \( t_{max,p} \) peers. If these conditions hold, then any peer in the DHT which executes the algorithms will select a peer uniformly at random.

### 3.2 Setting the Constants

To find candidate constant settings, we explored discrete points in a large space of possible constant values. Figure 3.2 shows the ranges 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 used the constant settings shown in Figure 8.

### 3.3 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 latency for a call to either algorithm is defined as \((\# \text{ of rounds} \cdot \log n) + \text{[\# of calls to next]}) \). To get the average latency we did the following: we generated 100 random DHTs. For each DHT, we performed 100 iterations of each algorithm. For each iteration, we selected a peer at random to perform the algorithm. We then computed the average latency over all 10,000 calls.

Figure 8 gives the settings chosen for the algorithms Peer Count and Arc Length respectively.

### 3.4 Empirical Results

Figure ?? shows the results of the tests. Each point represents an average of 10,000 executions of the algorithm: 100 trials on each of 100 random DHTs. For each trial, we selected a peer at random to perform the algorithm. We used the constant settings shown in Figure 8.

The results of the tests match the theoretical predictions. For Peer Count, the average number of rounds is about 5, with a latency of \( 19.1 \log n \). For Arc Length, the average number of rounds is about 4, with a latency of \( 11.4 \log n \). These results are for the specific constant values used in the tests. Since these constants are not necessarily optimal, it is not possible to compare them directly and conclude that one algorithm is preferable to the other. But it is clear that both algorithms carry small enough constant factors to be usable in practice.

### 4 Conclusion and Future Work

We have presented two new algorithms for choosing a peer uniformly at random from the set of all peers in a DHT. We have shown that these algorithms have good theoretical and empirical properties. Several open problems remain including: 1) generalizing these results to peer-to-peer networks which are less structured than DHTs and 2) generalizing these results to sensor networks.

### A Proofs of Correctness

In the following proofs, we will let \( f_i \) be a constant equal to \( 9,999/10,000 \).
A.1 Estimation algorithms

Lemma 3. For any $C, \epsilon > 0$, let $\xi_1(C, \epsilon)$ be the following event:

- There exists an interval $I$, such that $\text{len}(I) = (C \ln n)/n$ and $\text{num}(I) > (1 + \epsilon)C \ln n$.

Further let $b_1(C, \epsilon)$ equal:

$$ne^{-\frac{\epsilon^2}{2} \frac{(C \ln n)}{n}};$$

where $\delta = \frac{1+\epsilon}{n/n-\epsilon} - 1$ Then:

$$\text{Pr}(\xi_1(C, \epsilon)) \leq b_1(C, \epsilon).$$

Proof. The event we wish to bound is that there is some interval $I$ such that $\text{len}(I) = (C \ln n)/n$ and $\text{num}(I) > (1 + \epsilon)C \ln n$. If such an interval exists, there must also be an interval $I'$ such that $\text{len}(I') = (C \ln n)/n$ and $\text{num}(I') > (1 + \epsilon)C \ln n$ and the counterclockwise end starting at some peer point $p$. Thus, to bound the probability of $\xi_1(C, \epsilon)$, we need only consider those intervals whose counterclockwise end is a peerpoint.

Consider a fixed 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 value $\text{num}(I_p)$. By linearity of expectation, $E(X_p) = 1 + ((n-1)/n)(C \ln n)$.

Using Chernoff bounds, we can say that for all $\delta > 0$:

$$\text{Pr}[X_p > (1 + \delta)E(X_p)] < e^{-\frac{\delta^2 E(X_p)}{2 \ln n}}.$$

Setting $\delta = \frac{1+\epsilon}{n/n-\epsilon} - 1$, we get:

$$\text{Pr}[X_p > (1 + \epsilon)C \ln n] < e^{-\frac{\epsilon^2}{2} \frac{(C \ln n)}{n}}.$$

Now if we do a union bound over all peers $p$, we get that:

$$\text{Pr}(\xi_1(C, \epsilon)) \leq ne^{-\frac{\epsilon^2}{2} \frac{(C \ln n)}{n}} = ne^{-\frac{\epsilon^2}{2} \frac{(C \ln n)}{n}}.$$

Lemma 4. For any $C, \epsilon > 0$, let $\xi_2(C, \epsilon)$ be the following event:

- There exists an interval $I$, such that $\text{len}(I) = (C \ln n)/n$ and $\text{num}(I) < (1 + \epsilon)C \ln n.$

Further let $b_2(C, \epsilon)$ equal:

$$ne^{-\frac{\epsilon^2}{2} \frac{(C \ln n)}{n}};$$

where $\delta = 1 - (n/(n-1))(1 - \epsilon)$.

Then:

$$\text{Pr}(\xi_2(C, \epsilon)) \leq b_2(C, \epsilon).$$

Proof. For the event $\xi_2(C, \epsilon)$ to occur, there must be some interval $I$ such that $\text{len}(I) = (C \ln n)/n$ but $\text{num}(I) < (1 - \epsilon)C \ln n$. 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 $\xi_2(C, \epsilon)$, we need only consider such intervals.

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 value $\text{num}(I_p)$. By linearity of expectation, $E(X_p) = (\ln n - 1)/n(C \ln n)$. Using Chernoff bounds, we can say that for all $\delta > 0$:

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

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

$$\text{Pr}[X_p < (1 - \epsilon)C \ln n] < e^{-\frac{\epsilon^2}{3} \frac{(C \ln n)}{n}}.$$

Now if we do a union bound over all peers $p$, we get that:

$$\text{Pr}(\xi_2(C, \epsilon)) \leq ne^{-\frac{\epsilon^2}{3} \frac{(C \ln n)}{n}} \leq e^{\ln n - \frac{\epsilon^2}{3} \frac{(C \ln n)}{n}}.$$

Lemma 5. For any $C, \epsilon > 0$, let $\xi_3(C, \epsilon)$ be the following event:

- There exists an interval $I$, such that $\text{len}(I) = (C \ln n)/n$ and $\text{num}(I) < (1 - \epsilon)C \ln n.$

or $\text{num}(I) > (1 + \epsilon)C \ln n.$
Further let \( b_3(C, \epsilon) \) equal:

\[
b_1(C, \epsilon) + b_2(C, \epsilon).
\]

Then:

\[\Pr(\xi_3(C, \epsilon)) \leq b_3(C, \epsilon)).\]

**Proof.** The proof is immediate from Lemmas 3 and 4 and the use of a union bound. \( \square \)

**Lemma 6.** For any \( C, \epsilon > 0 \), let \( \xi_4(C, \epsilon) \) be the following event:

- There exists an interval \( I \), such that \( \text{num}(I) = C \ln n \) and
  \[
  \text{len}(I) \leq (1 - \epsilon)(C \ln n)/n.
  \]

Further let \( b_4(C, \epsilon) \) equal:

\[
b_1((1 - \epsilon)C, \epsilon/(1 - \epsilon)))
\]

Then:

\[\Pr(\xi_4(C, \epsilon)) \leq b_4(C, \epsilon)).\]

**Proof.** Consider the event that there is an interval \( I \) such that \( \text{num}(I) = (C \ln n) \) and \( \text{len}(I) \leq (1 - \epsilon)(C \ln n)/n \). For such an event to occur, there must be an interval \( I' \) such that \( \text{len}(I') = (1 - \epsilon)(C \ln n)/n \) and \( \text{num}(I') \geq (C \ln n) \). By definition, this is the event \( \xi_4((1 - \epsilon)C, \epsilon/(1 - \epsilon)) \). \( \square \)

**Lemma 7.** For any \( C, \epsilon > 0 \), let \( \xi_5(C, \epsilon) \) be the following event:

- There exists an interval \( I \), such that \( \text{num}(I) = C \ln n \) and
  \[
  \text{len}(I) \leq (1 - \epsilon)(C \ln n)/n.
  \]

Further let \( b_5(C, \epsilon) \) equal:

\[
b_2((1 + \epsilon)C, \epsilon/(1 + \epsilon))
\]

Then:

\[\Pr(\xi_5(C, \epsilon)) \leq b_5(C, \epsilon)).\]

**Proof.** Consider the event that there is an interval \( I \) such that \( \text{num}(I) = (C \ln n) \) and \( \text{len}(I) \geq (1 + \epsilon)(C \ln n)/n \). For such an event to occur, there must be an interval \( I' \) such that \( \text{len}(I') = (1 + \epsilon)(C \ln n)/n \) and \( \text{num}(I') \leq (C \ln n) \). By definition, this is the event \( \xi_2((1 + \epsilon)C, \epsilon/(1 + \epsilon)) \). \( \square \)

**Lemma 8.** For any \( C, \epsilon > 0 \), let \( \xi_6(C, \epsilon) \) be the following event:

- There exists an interval \( I \), such that \( \text{num}(I) = C \ln n \) and
  \[
  \text{len}(I) \leq (1 - \epsilon)(C \ln n)/n.
  \]
  or
  \[
  \text{len}(I) \geq (1 + \epsilon)(C \ln n)/n.
  \]

Further let \( b_6(C, \epsilon) \) equal:

\[
b_1((1 - \epsilon)C, \epsilon/(1 - \epsilon))) + b_2((1 + \epsilon)C, \epsilon/(1 + \epsilon))
\]

Then:

\[\Pr(\xi_6(C, \epsilon)) \leq b_6(C, \epsilon)).\]

**Proof.** The proof is immediate by a union bound on Lemmas 6 and 7. \( \square \)

**Lemma 9.** Let \( \alpha_1 < 1 \) and \( \alpha_2 > 1 \) be fixed constants and let \( C \) be the constant used in Procedure 1. Let \( \xi_7(\alpha_1, \alpha_2) \) be the following event:

- For some peer, \( p \), Procedure 1 returns an estimate which is either smaller than \( \alpha_1 \ln n \) or larger than \( \alpha_2 \ln n \).

Further let \( b_7(\alpha_1, \alpha_2) = e^{C(n - n^{1 - \alpha_1} + C + 1 + \ln n + e^{\ln n + C(1 - \alpha_2) \ln n + C}} \)

Then:

\[\Pr(\xi_7(\alpha_1, \alpha_2)) \leq b_7(\alpha_1, \alpha_2).\]

**Proof.** Consider the event that some peers estimate is smaller than \( \alpha_1 \ln n \). For this to happen, it must be the case that for some peer, \( p \), less than \( C \) peers fall in the interval \( (p, p + C/n^{\alpha_1}) \). For this to occur, at least \( n - C \) peers must fall in an interval of size \( 1 - C/n^{\alpha_1} \). This probability is no more than:

\[
\left( \frac{n}{n - C} \right) (1 - C/n^{\alpha_1})^{n - C} = \left( \frac{n}{C} \right) (1 - C/n^{\alpha_1})^{n - C} \leq (ne/C)^C e^{-(n-C)/n^{\alpha_1}} = e^{C(n + 1 - n^{1 - \alpha_1} + C + C/n^{\alpha_1})}
\]

Taking a union bound over all \( n \) peers, we have that the probability that any peer’s estimate is smaller than \( \alpha_1 \ln n \) is no more than:
Consider the event that some peers estimate is larger than \(\alpha_2 \ln n\). For this to happen, there must be some peer, \(p\), such that \(C\) peers fall in the interval \((p, p + C/n^{\alpha_2})\). The probability of this event is no more than

\[
\binom{n}{C} \left(\frac{C}{n^{\alpha_2}}\right)^{C} \leq \frac{e^{n+C} (n^{1-\alpha_2} + C/n^{\alpha_2})}{e^{n} (n^{1-\alpha_2})^{n} + C/n^{\alpha_2}}
\]

Taking a union bound over the \(n\) peers, we have that the probability that any peer’s estimate is larger than \(\alpha_2 \ln n\) is no more than:

\[
ne^{C(1-\alpha_2)\ln n + C} = e^{n+C(1-\alpha_2)\ln n + C}
\]

Lemma 10. The expected total latency of Peer Count is no more than:

\[
C_1 + C_2s_p + (t_p/(ndmin_p))(\ln n + t_p)
\]

Proof. Note that the total latency of FindParametersI is:

\[
C_1 + C_2s_p.
\]

We can bound the expected latency of the rest of Peer Count by noting that: 1) the expected number of rounds is \(t_p/ndmin_p\) and 2) the latency of each round is no more than \(\ln n + t_p\) (this includes at most \(\ln n\) latency for the call to \(h()\) and \(t_p\) latency for at most \(t_p\) calls to \(next()\)). Thus the total expected latency of Peer Count is no more than:

\[
C_1 + C_2s_p + (t_p/(ndmin_p))(\ln n + t_p)
\]

Theorem 11. Let \(\epsilon_1, \epsilon_2, \alpha_1, \alpha_2, C_1, C_2, C_4\) be any positive constants and let \(C_3 = (1 - \epsilon_2)/(1 - \epsilon_1)\). Then Peer Count is correct and has expected latency no more than:

\[
C_1 + C_2\alpha_2 \ln n + [\alpha_2/(\alpha_1(1 - \epsilon_2))][C_4(\alpha_2 + 1)\ln n].
\]

with probability of error no more than:

\[
b_7(\alpha_1, \alpha_2) + b_6(C_2\alpha_1, \epsilon_1) + b_4(C_4\alpha_1, \epsilon_2).
\]

Proof. We will bound the probability that this algorithm fails by bounding the probabilities that different steps of the algorithm fail.

Consider the following three events:

- \(\chi_1\) is the event that \(s_p = \alpha \ln n\) for some \(\alpha_1 \leq \alpha \leq \alpha_2\).
- \(\chi_2\) is the event that:

\[
(1 - \epsilon_1)s_p/n \leq L_p \leq (1 + \epsilon_1)s_p/n.
\]

- \(\chi_3\) is the event that for all intervals, \(I\) such that \(num(I) = t_p\), it must be the case that \(len(I) \geq dmin_p\).

We now bound the probability that any of these three events fail to occur. Note that:

\[
Pr(\chi_1 \cup \chi_2 \cup \chi_3) = Pr(\chi_1 \cup \chi_2 \cup \chi_3) \leq Pr(\chi_1) + Pr(\chi_2) + Pr(\chi_3).
\]

We first note that \(P(\chi_1) \leq b_7(\alpha_1, \alpha_2)\) by Lemma 9. Second, note that the event, \(\chi_2\), is equivalent to the event that there is an interval, \(I\), such that \(num(I) = C_2\alpha \ln n\) and \(len(I) < (1 - \epsilon_1)(C_2\alpha \ln n)/n\) or \(len(I) > (1 + \epsilon_1)(C_2\alpha \ln n)/n\). This is just the event \(\xi_6(C_2\alpha, \epsilon_1)\). Thus by Lemma 8:

\[
P(\chi_2) \leq b_6(C_2\alpha_1, \epsilon_1) + b_4(C_4\alpha_1, \epsilon_2).
\]

Finally note that the event \(\chi_3\), is equivalent to the event that there exists an interval, \(I\), such that \(num(I) = C_4\alpha \ln n\) but \(len(I) < C_3C_4(1 - \epsilon_1)\alpha \ln n\). Since \(C_3 = (1 - \epsilon_2)/(1 - \epsilon_1)\), this is just the event \(\xi_4(C_4\alpha, \epsilon_2)\). Thus by Lemma 6,

\[
P(\chi_3) \leq b_4(C_4\alpha_1, \epsilon_2).
\]

This implies that one of the events \(\chi_1, \chi_2\) and \(\chi_3\) fail to occur with probability no more than:

\[
b_7(\alpha_1, \alpha_2) + b_6(C_2\alpha_1, \epsilon_1) + b_4(C_4\alpha_1, \epsilon_2).
\]

We can choose constants \(C_1, C_3\) and \(C_4\) to ensure that this probability is no more than \(1/n\). Note that if all of these events occur, the following facts are true:

- For all intervals, \(I\) such that \(num(I) = t_p\), it is the case that \(len(I) \geq dmin_p\) (i.e. Peer Count is correct);
- \(s_p \leq \alpha_2 \ln n\);
- \(d_p \geq (1 - \epsilon_1)(\alpha_1 \ln n)/n\)
The expected total latency of Arc Length is no more than:

\[ C_1 + C_2 \alpha_2 \ln n + (\alpha_2/(\alpha_1(1 - \epsilon_2)))((C_4 \alpha_2 + 1) \ln n). \]

**Lemma 12.** The expected total latency of Arc Length is no more than:

\[ C_1 + C_2 t_p + (t_{max}/(nd_p)) \ln n + t_{max} p \]

**Proof.** Note that the total latency of FindParameters is:

\[ C_1 + C_2 t_p. \]

We can bound the expected latency of the rest of Arc Length by noting that: 1) the expected number of rounds is \((t_{max}/nd_p)\) and 2) the expected latency of each round is no more than \(\ln n + nd_p\). Thus the total expected latency of Arc Length is no more than:

\[ C_1 + C_2 t_p + (t_{max}/(nd_p)) \ln n + t_{max} p \]

**Theorem 13.** Let \(\epsilon_1, \epsilon_2, \alpha_1, \alpha_2, C_1, C_2, C_4\) be any positive constants and let \(C_3 = (1 + \epsilon_2)(1 + \epsilon_1)\). Then Arc Length is correct and has expected latency no more than:

\[ C_1 + C_2 \alpha_2 \ln n + [(C_3 \alpha_2)/((1-\epsilon_1)\alpha_1)] \ln n + C_4 C_3 \alpha_2 \ln n \]

with probability of error no more than:

\[ b_7(\alpha_1, \alpha_2) + b_0(C_2 \alpha_1, \epsilon_1) + b_1(C_4(1 + \epsilon_1)\alpha_2, \epsilon_2). \]

**Proof.** We will again bound the probability that this algorithm fails by bounding the probabilities that different steps of the algorithm fail.

Consider the following three events:

- \(\chi_1\) is the event that \(t_p = \alpha \ln n\) for some \(\alpha_1 \leq \alpha \leq \alpha_2\).
- \(\chi_2\) is the event that:
  \[(1 - \epsilon_1)t_p/n \leq d_p \leq (1 + \epsilon_1)t_p/n.\]
- \(\chi_3\) is the event that for all intervals, \(I\) such that \(\text{len}(I) = d_p\), it is the case that \(\text{num}(I) \leq t_{max}\).

We now bound the probability that any of these three events fail to occur. Note that:

\[ \Pr(\chi_1) \leq b_7(\alpha_1, \alpha_2) \]

Our first note that \(\Pr(\chi_1) \leq b_7(\alpha_1, \alpha_2)\) by Lemma 9. Second, note that the event, \(\chi_2\), is equivalent to the event that there is an interval, \(I\), such that \(\text{num}(I) = C_2 \alpha \ln n\) and \(\text{len}(I) < (1 - \epsilon_1)(C_2 \alpha \ln n)/n\) or \(\text{len}(I) > (1 + \epsilon_1)(C_2 \alpha \ln n)/n\). This is just the event \(\xi_6(C_2 \alpha, \epsilon_1)\). Thus by Lemma 8:

\[ \Pr(\chi_2|\chi_1) \leq b_6(C_2 \alpha_1, \epsilon_2). \]

Finally note that the event, \(\chi_3|\chi_2, \chi_1\), is equivalent to the event that there exists an interval, \(I\), such that \(\text{len}(I) = C_4(1 + \epsilon_1)(\alpha \ln n)/n\) but \(\text{num}(I) > C_4 C_3 \alpha \ln n\). Since \(C_3 = (1 + \epsilon_2)(1 + \epsilon_1)\), this is just the event \(\xi_1(C_4(1 + \epsilon_1)\alpha, \epsilon_2)\). Thus by Lemma 6,

\[ \Pr(\chi_3|\chi_2, \chi_1) \leq b_1(C_4(1 + \epsilon_1)\alpha_1, \epsilon_2). \]

This implies that one of the events \(\chi_1\), \(\chi_2\) and \(\chi_3\) fail to occur with probability no more than:

\[ b_7(\alpha_1, \alpha_2) + b_6(C_2 \alpha_1, \epsilon_1) + b_1(C_4(1 + \epsilon_1)\alpha_2, \epsilon_2). \]

We can choose constants \(C_2, C_3\) and \(C_4\) to ensure that this probability is no more than \(1/n\). Note that if all of these events occur, the following facts are true:

- For all intervals, \(I\) such that \(\text{len}(I) = d_p\), it is the case that \(\text{num}(I) \leq t_{max}\) (i.e. Arc Length is correct);.
- \(t_p \leq \alpha_2 \ln n;\)
- \(t_{max} \leq C_4(1 + \epsilon_2)(1 + \epsilon_1)\alpha_2 \ln n\)
- \(d_p \geq C_4(1 - \epsilon_1)(\alpha_1 \ln n)/n\)

By Lemma 12, the total latency of Arc Length is no more than:

\[ C_1 + C_2 t_p + (t_{max}/(nd_p)) \ln n + t_{max} p \]
Plugging in the facts above, we can say that the expected number of rounds of Arc Length is no more than \[((1 + \epsilon_2)(1 + \epsilon_1)\alpha_2)/((1 - \epsilon_1)\alpha_1)\). Thus the expected latency is no more than:

\[C_1 + C_2\alpha_2 \ln n + [(1+\epsilon_2)(1+\epsilon_1)\alpha_2]/((1-\epsilon_1)\alpha_1)]\ln n + C_4(1+\epsilon_2)(1+\epsilon_1)\alpha_2 \ln n\]