RootChord

by

Lukasz Cwik

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2010

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

In this thesis, we present a distributed data structure, which we call "RootChord". To our knowledge, this is the first distributed hash table which is able to adapt to changes in the size of the network and answer lookup queries within a guaranteed two hops while maintaining a routing table of size $\Theta(N^{\frac{1}{2}})$. We provide pseudocode and analysis for all aspects of the protocol including routing, joining, maintaining, and departing the network. In addition we discuss the practical implementation issues of parallelization, data replication, remote procedure calls, dead node discovery, and network convergence.

Acknowledgements

I would like to thank my supervisor Ian Munro and colleague Bryan Logan for all the discussions about distributed hash tables we had over the past few months.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

A distributed hash table (DHT) is a peer-to-peer (P2P) data structure that implements $\{key, value\}$ based storage and retrieval operations. This data structure has been heavily studied since the introduction of what was considered the first DHT by Plaxton *et al.* [55] over a decade ago. Initial DHT data structures such as Fasttrack derivatives (Gnutella [3] and Kazaa [5]) had a best effort query mechanism which could not guarantee a successful lookup even if the *key* was in the network. The release of DHTs such as Chord [62] with guaranteed lookups ignited much more research in this field. Since Chord, many desirable properties of DHTs have been discovered and discussed. Minimally, we require that a DHT must be able to insert a $\{key, value\}$ pair at a peer in such a way that a future lookup query for the *key* will succeed. Other desirable properties include:

1. *Fault tolerance:* The structure should adjust to the failure of some nodes and stale data should not hamper a system from functioning.

2. *Robustness:* Queries should always be able to be routed even under heavy churn

within the network.

3. *Fast queries:* The structure should not have to contact many nodes to lookup a $\{key, value\}$ pair. Also, a minimal number of messages should be exchanged in order to complete the requested operation. Internal computation time at hosts is not counted as message delays are the typical efficiency bottleneck.

4. *Different deployment environments:* A DHT should adapt dynamically to the size of the network and have parameters which can be used to fine tune its operation for specific scenarios.

5. *Parallelizable:* All aspects of the DHT should be parallelizable to be able to avoid high latency / low bandwidth nodes.

6. *Homogeneous nodes:* Each node should behave as every other node and should have an equal distribution of work. Hence the congestion per node should be equal.

From this list of properties, we have created a new DHT which we call "RootChord". This new DHT is a non-trivial adaptation of Chord by Stoica *et al.* [62]. We begin by delving into previous work with an explanation of Chord and other relevant DHTs. We then move onto the protocol in Chapter 2 and discuss extensions in Chapter 3. Finally, we conclude in Chapter 4.

1.1 Previous Work

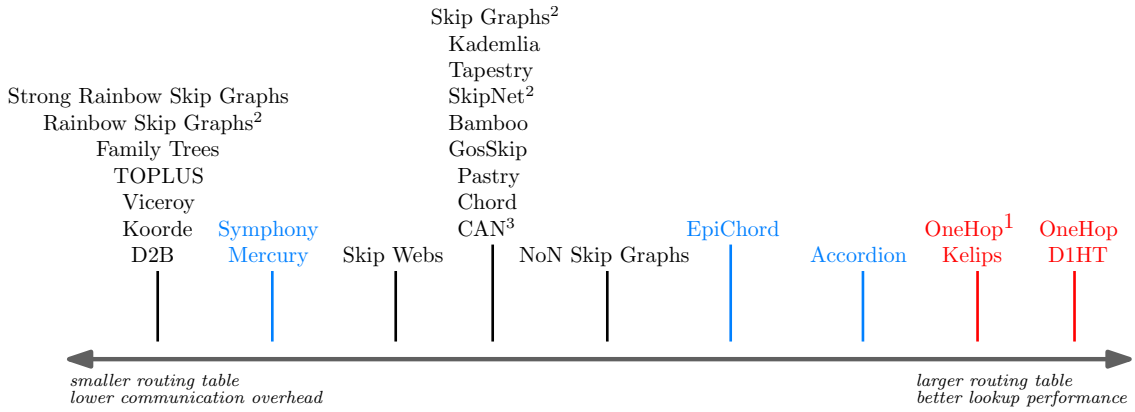
The introduction of DHTs such as CAN [57], Chord [62], Pastry [60], and Tapestry [64] in 2001 started a frenzy of research within the field of DHTs. The research progressed within

four directions, the first being the development of new features on top of existing DHTs. One example of this is to allow for ordered data within the DHT as is done in Rainbow Skip Graphs [23]. The structure supports queries that are based on an ordering, extending DHTs from being only dictionaries capable of answering membership queries to being able to perform nearest-neighbour searches and range queries.

At the same time, research progressed into looking at ways to improve the practical performance of DHTs. Many studies profiled existing DHTs and concluded that issues arise due to churn in the network and heterogeneity of the nodes. Rhea *et al.* [59] studied the effects of slow nodes within OpenDHT (an implementation of Bamboo [59]) within PlanetLab [8] and concluded that usage of delay aware routing was needed. This meant that nodes should store round trip time information for peers and adapt the routing technique to select peers which would respond quickly. Similarly, Falkner *et al.* [20] profiled one million users using Azureus's [1] DHT (a modified version of Kademlia [47]) and noticed that half the nodes were inactive after 1 hour, while only 5% percent remained after 48 hours. The unpredictability of this environment leads to heavy tailed session times, inconsistent routing tables and high overhead even though the operation was robust, leveraging data replication and routing redundancy.

While research progressed to improve the feature sets and real world practical performance of DHTs, theoretical bounds were improved for decreasing the size of the routing table, and also decreasing the cost of a lookup, but only one at the expense of the other. The lookup cost is measured by the amount of hops required to reach the destination. A **hop** is considered to be the traversal of an edge joining two nodes within a network. Generally, DHTs are categorized based on their lookup costs. There are those with logarithmic

based hop counts such as GosSkip [24], SkipNet [29], and Viceroy [44]. Then there are those with variable hop counts with expected bounds such as Accordion [41], Symphony [45], and Mercury [15]. Finally there are those with large routing tables but guaranteed hop counts which are $\mathcal{O}(1)$ such as Kelips [27], OneHop [26], and D1HT [51] (also known as ERDA).



Lookup Hopcount	$\mathcal{O}(\log N)$	$\mathcal{O}'(\frac{\log^2 N}{m})$	$\mathcal{O}'(\frac{\log N}{\log\log N})$	$\mathcal{O}(\log N)$	$\mathcal{O}'(\frac{\log N}{\log\log N})$	$\mathcal{O}'(\frac{\log N}{\log j})$	$\mathcal{O}'(\frac{\log N \log\log N}{\log k})$	2	1
Routing Table Size	$\mathcal{O}(1)$	$\mathcal{O}(m)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(\log^2 N)$	$\mathcal{O}(j \log N)$	$\mathcal{O}(k)$	$\mathcal{O}(N^{\frac{1}{2}})$	$\mathcal{O}(N)$

Notes: $\mathcal{O}'$ represents an expected bound
$j \ll N$
$k \geq \log N$
$m \leq \log N$
1 OneHop also provides a two hop DHT
2 Lookup hopcount with high probability
3 CAN is in the $\mathcal{O}(\log N)$ group even though it is $\mathcal{O}(dN^{\frac{1}{d}})$ with table size $\mathcal{O}(d)$. Since its minimum is when $d = \log N$ and there are already other DHTs with routing tables of constant size.

Figure 1.1: Existing DHTs Summary

Figure 1.1 shows a layout of many DHTs and their grouping. Black DHTs represent research in logarithmic based hop routing, blue DHTs represent work in variable hop count routing, while red DHTs represent work towards constant hop routing. Since we are creating a DHT which falls into the 2 hop count lookups category, it is important to consider the DHTs which have similar capabilities to what we have created. Thus, below

we start with an explanation of Chord which we base our DHT on, and then proceed to discuss theoretically similar DHTs such as Kelips and OneHop.

1.1.1 Chord

Chord [62] is a well known DHT with a ring shaped structure and b-bit numerical identifiers. Chord performs arithmetic operations modulo 2^b and the identifier ring is represented as the integers from 0 to $2^b - 1$. Keys are chosen from this identifier ring. The **successor** of a key k is the first node with identifier greater than or equal to k while the **predecessor** is the first node with an identifier strictly less than k. Each node is responsible for all the keys which they are the successor of. In other words, a node is responsible for all the keys between its predecessor and itself.

The routing information or **finger table** maintained by each node contains information about the nodes that are successively further apart by a power of 2 around the identifier ring. Hence, a node A knows of all the nodes that are successors of $A+2^{i-1}$ where $1 \leq i \leq b$. In addition, each node knows of its predecessor. Thus the finger table contains $b+1$ entries. In practice many of these entries are duplicates which leads to the finger table containing $\mathcal{O}(\log N)$ unique entries. These duplicates exist because the expected distance between two nodes is $\mathcal{O}(\frac{N}{\log N})$. Thus, for values of i near 1, it is highly probable that the successors of $A + 2^{i-1}$ are not unique.

Maintenance of the finger table is handled by a background process which regularly looks up the successor of $A+2^{i-1}$ for $1 \leq i \leq b$. The time between periodic maintenance is chosen by the application. If the periodic maintenance occurs often, then Chord's routing table is more efficient at the cost of more traffic.

Routing is done in logarithmic time. This is accomplished by routing to the nearest finger less than or equal to the destination value. Since the distance between nodes in the finger table are successors of powers of 2, each hop reduces the distance to the target node approximately by half.

Replication is application specific but practice has been to store replicas in the r nodes succeeding the key's identifier. Larger values of r increase the probability that information survives node failures but also increases the probability for stale data since many peers must be updated. The value of r is application specific.

When a Chord node joins, it chooses the SHA-1 hash of its IP address as its identifier. The node then proceeds to lookup its own identifier treating the result as its successor and the first finger table entry. The node then searches for its predecessor and additional fingers. When a node departs, the leaving node informs its predecessor and successor and transfers the $\{key, value\}$ pairs to the successor.

1.1.2 Variable Hop Count Lookups

Kleinberg studied the small world phenomenon [37], the principle that we are all linked by short chains of acquaintances, also known as the "six degrees of separation" phenomenon. He discovered that one is able to construct a searchable network by using long-range links whose probabilities decay with distance. This has proved useful in the design of peer-to-peer file-sharing systems such as Accordion, Symphony, Mercury, and EpiChord which all rely on this phenomenon.

For example, Accordion uses the small worlds distribution to choose its neighbours: the probability of a node selecting a neighbour with distance d in the identifier space from

itself is proportional to $\frac{1}{d}$. This distribution causes a node to prefer neighbours that are closer to itself in the identifier space, ensuring that as a lookup gets closer to the target key there is likely to be a helpful routing table entry.

1.1.3 One Hop Lookups

One hop DHTs rely on total world knowledge to be able to answer lookup queries. Thus this realm is not limited to only DHTs but routing protocols such as Open Shortest Path First (OSPF) [7] and Interior Gateway Routing Protocol (IGRP) [4], or epidemic [13, 18] and gossip [36, 10] protocols. For example D1HT uses an Event Detection and Report Algorithm (ERDA for short) to disseminate messages in $\mathcal{O}(\log n)$ time by dividing the work up among peers. Figure 1.2 shows an example of ERDA in action when peer P needs to deliver a message to all peers. P would send it to all peers with a distance of 2^i away from itself. These peers at a distance of 2^i would then send messages again to all peers up to 2^{i-1}.

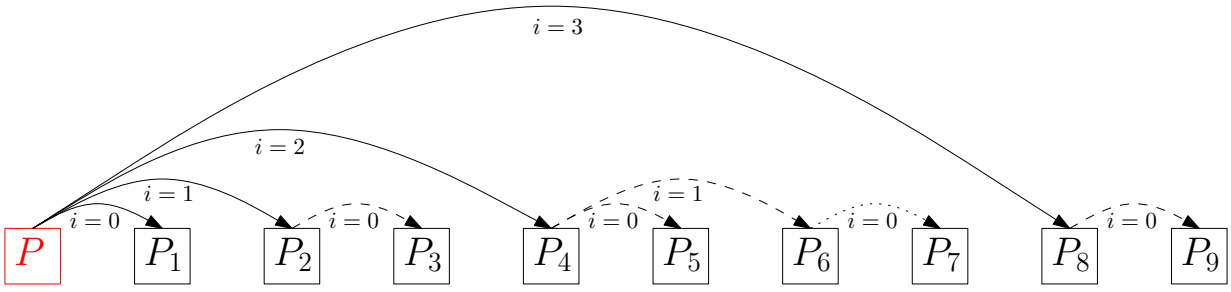


Figure 1.2: Event Propagation in D1HT

1.1.4 Two Hop Lookups

The small world distribution found in the variable hop DHTs such as EpiChord result in lookups with expected bounds, and hence cannot guarantee that a lookup will succeed with only 2 hops. While the one hop lookups are based on total world knowledge of every other node and hence carry the routing table cost of $\mathcal{O}(N)$. Thus, we will now look at the only two DHTs to our knowledge with a lookup cost of 2 hops. These are Kelips [27] and a variant of OneHop [26]. We will see that both of these schemes suffer from being unable to scale with changes in the network size.

Kelips

Kelips is a group based DHT that disseminates information about changes in a group by using a lightweight epidemic multicast protocol [36] for replicating system membership and file tuple data. Kelips consists of k groups numbered 0 through $(k-1)$ and each node lies in a group determined by using a consistent hashing [34, 39] function. This function maps the node's identifier (IP address and port number) to an integer in $[0, k-1]$. Each node stores a partial set of other nodes lying in the same group and a constant number of nodes for every other group. For a $\{key, value\}$ pair to be inserted, the key is mapped into a group and an insert request is sent to the closest known contact for that group. The contacted node chooses a node at random from within its group as the *homenode* for the value. The key and *homenode* are inserted into the gossip stream and disseminated to the entire group.

Thus for a node to look up a key, it uses the consistent hashing function to map the key into one of the k groups. The node then contacts the topologically closest known contact

for that group which replies with the *homenode* for said *key*. Figure 1.3 shows an example of node 12 in group 0 looking up a key which hashes to group 1. Node 12 contacts node 13 asking for the $\{key, value\}$ pair. Node 13 responds with the address of the *homenode* 17.

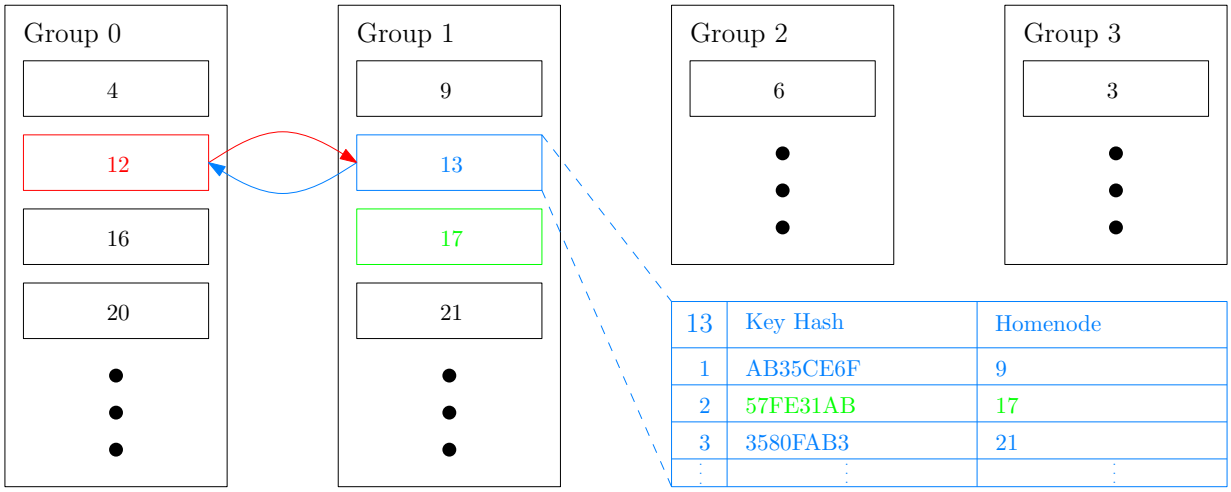


Figure 1.3: Kelips Lookup Protocol

When $k = \mathcal{O}(N^{\frac{1}{2}})$, Kelips is able to maintain a routing table of size $\mathcal{O}(N^{\frac{1}{2}})$ per node. But Kelips was designed with the value k to be chosen during network creation. Hence the routing table is really $\mathcal{O}(\max(\frac{N}{k}, k))$ and a poor value for k during network creation will greatly affect the size of the routing table for each node within the network. Thus Kelips is unable to handle deployment environments where the network size changes over time.

OneHop Variant

OneHop was originally designed to facilitate a one hop lookup scheme but was adapted to provide a two hop lookup scheme for larger networks. Here we provide a description of the two hop scheme.

Every node in the overlay is assigned a random 128-bit node identifier. Identifiers are ordered in an identifier ring modulo 2^{128}. We assume that identifiers are generated such that the resulting set is uniformly distributed in the identifier space, for example, by setting a node's identifier to be the cryptographic hash of its network address. Every node has a predecessor and a successor in the identifier ring, and it periodically sends keep-alive messages to these nodes.

Similarly, each item has a *key*, which is also an identifier in the ring. The mapping from keys to nodes is based on the one used in Chord [62]. Hence, responsibility for an item rests with its successor.

The identifier ring is divided into k equal size slices, and one node is tasked with being the **slice leader**. The **slice leader** is responsible for disseminating changes within its own group. Each node within a group knows of every other node within the same group and one node for every other group.

When $k = \mathcal{O}(N^{\frac{1}{2}})$, OneHop is able to maintain a routing table of size $\mathcal{O}(N^{\frac{1}{2}})$ per node. And similarly to Kelips, a OneHop node's routing table is really $\mathcal{O}(\max(\frac{N}{k}, k))$. In addition to this fault, OneHop also suffers from having the **slice leader** becoming a bottleneck since all the nodes within the group rely upon it.

Chapter 2

A New Protocol: RootChord

This section describes a new protocol which is similar to Chord [62] in that it uses the same identifier ring modulo 2^b and data replication scheme. The protocols, however, differ in most other aspects.

In this section we discuss how to find the location of keys, join the network, maintain the network, leave the network gracefully, and detect node failures. We assume that the underlying network is both symmetric (if A can communicate directly with B, then B can communicate directly with A) and transitive (if A can communicate with B and B can communicate with C, then A can communicate with C).

2.1 Overview

RootChord uses a b-bit identifier space where 2^b is much larger than the maximal size of the network. Identifiers are ordered in an identifier ring modulo 2^b. Nodes do not generate their own identifiers at random but choose an identifier from a set of possible identifiers during

the bootstrap process. $\{key, value\}$ pairs are stored at the first node with an identifier greater than or equal to the key.

2.1.1 Notation and Conventions

The identifier ring is numbered from 0 to $2^b - 1$ and all arithmetic is performed modulo 2^b within this ring. Commonly, we refer to A and B as nodes, k as a key, and use the terms **finger table** and routing table interchangeably. If we wish to specifically give a node A or a key k a specific identifier, then we will subscript them. Hence A_{40} is the node A with identifier 40 and k_{25} is the key k with identifier 25.

If $B \in (A, A+2^{b-1}]$, then we say that B is to the right of A, otherwise if $B \in (A-2^{b-1}, A)$ then B is to the left of A. We also state that $A < B$ if A is to the left of B and similarly $A > B$ if A is to the right of B. The LEFT/RIGHT neighbour of a node A, is the first node which is to the left/right of A. The successor of a key k is A_k and, if no such node exists, then it is the first node to the right of k. Similarly, the predecessor is the first node to the left of k. The distance between two identifiers A and B is the length of the shortest path from A to B around the identifier ring. The distance is defined as $\text{MIN}(\boldsymbol{A - B \bmod 2^b, B - A \bmod 2^b})$ and is represented as $A - B$. A gap is considered to be the distance between two adjacent nodes A and B. The size of a gap is defined as $A - B$ and without loss of generality, if $A \leq B$, then its midpoint is $A + \left\lfloor \frac{B-A}{2} \right\rfloor$.

Finally, N is the size of the network, and N_A is A's estimate of the size. α is the distance from which a node is responsible for to be able answer successor queries. Specifically, α_A is node A's estimate for α, and hence A is able to answer any successor query in the range $[A - \alpha_A, A + \alpha_A]$.

2.1.2 Function Definitions

A.FunctionCall(...) is a remote procedure call to node A for procedure FunctionCall(...). If the node is unspecified as in FunctionCall(...), it is assumed to represent a procedure call on the local node. Observe that we are overloading function definitions below.

1. Random(R) returns an integer distributed uniformly at random within the discrete range R.
2. Random(S) returns an element distributed uniformly at random from the set S.
3. Finger(k) finds the node in the finger table with least distance to k.
4. Finger(R) finds all nodes in the discrete range R in the finger table.
5. Successor(k) finds the node in the finger table which is the successor of k.
6. Successor(R) finds all nodes in the finger table which are successors of any value in the discrete range R.
7. Predecessor(k) finds the node in the finger table which is the predecessor of k.
8. Midpoint(G) finds the midpoint for a gap G.
9. MinGap(R) returns the smallest gap between all adjacent nodes in the discrete range R.
10. MaxGap(R) returns the largest gap between all adjacent nodes in the discrete range R.

2.2 Routing

Routing in a network consists of an understanding of the constraints on the topology of the network and the knowledge of the links which nodes know. In this section we start by describing the contents of the finger table and then move on to show how a node estimates the size of the network. We conclude with a proof of the bound on the number of nodes within the finger table and discuss how a node would lookup a key.

2.2.1 Finger Table

The finger table for a node within this network is divided into two areas: a set of **local peers**, and a set of **distant peers**.

Local peers are defined as the peer set required to directly answer any successor queries within a distance of α. Hence, for a node A, these are all the peers within the range $[A-\alpha_A, A+\alpha_A]$ plus the successor of $A+\alpha_A$. Thus if a query is ever made for a key $k \in [A-\alpha_A, A+\alpha_A]$, then A should be able find the successor of k in its own cache of peers without contacting another node.

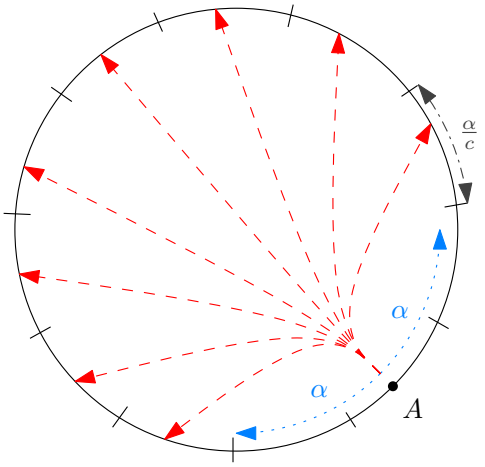


Figure 2.1: Graphic representation of a finger table

Distant peers are defined as the set of peers that are required to be contacted to answer any other successor query. There is a bound on the distance between adjacent distant peers. It is defined as $\frac{2\alpha}{c}$ for some constant $c \geq 1$, usually we will take $c = \sqrt{2}$. The

constant c is used to guarantee a lookup cost of 2 hops and is discussed further in Section 2.6.

Figure 2.1 shows an example describing the ranges defining the local and distant peer sets. The range described for A's local peers is covered by the *blue dotted* arrows, while the *red dashed* lines represent A's distant peers. One immediately sees that when these two ranges are combined, the entire address space is covered.

2.2.2 Network Size Estimate

Estimating the network size is key to control α. α is used to bound the space for which a node is responsible for to answer successor queries within. Also, by definition, α is used to differentiate between local and distant peers. The parameter α satisfies the following equation

$$\alpha = \frac{2^b}{N^{\frac{1}{2}}}. \tag{2.1}$$

To compute α, one must know the size of the network N. But keeping the number N accurately among all nodes in the network is too costly. Thus, a node A uses an approximation algorithm to compute N_A and α_A, A's estimates of the network size N and α. This approximation algorithm assumes that the number of nodes in the region $[A - \alpha_A, A + \alpha_A]$ is $2N^{\frac{1}{2}}$. In other words we assume that

$$2N_A^{\frac{1}{2}} = |\textsc{Finger}([A - \alpha_A, A + \alpha_A])|$$

and thus we may obtain the approximation

$$\alpha_A = \frac{2^{b+1}}{|\textsc{Finger}([A - \alpha_A, A + \alpha_A])|}.$$

In general the approximation algorithm goes as follows: a node A begins with $\alpha_A = 0$. A continuously increases α_A while

$$\alpha_A \cdot |\textsc{Finger}([A - \alpha_A, A + \alpha_A]| < 2^{b+1}$$

is satisfied. Below we provide pseudocode to the approximation algorithm and in Section 2.2.3 we prove bounds on the error in the network size and α.

Pseudocode

In the pseudocode of Algorithm 1, lines [5-6] initialize the left and right nodes to A. Line 7 checks that the bound is satisfied. Lines [8-15] expand our local peer range depending on whether *left* or *right* is closer. In the case of a tie, we choose *right*. Lines [8-15] may have overshot our α_A estimate by adding one too many nodes into our local peer range. Thus, if a smaller α_A still satisfies the constraint on line 16 we compute the new α_A on line 17.

Algorithm 1 Computing α

Preconditions:
A is the current network node.

ComputeAlpha()
$left \Leftarrow A$
$right \Leftarrow A$
while $\alpha_A \cdot |\textsc{Finger}([A - \alpha_A, A + \alpha_A]| < 2^{b+1}$ **do**
  if $right - A \leq A - left$ **then**
    $right \Leftarrow right.\textsc{Right}$
    $\alpha_A \Leftarrow right - A$
  else
    $left \Leftarrow left.\textsc{Left}$
    $\alpha_A \Leftarrow A - left$
  end if
end while
if $(\alpha_A - 1) \cdot (|\textsc{Finger}([A - (\alpha_A - 1), A + (\alpha_A - 1)]|) > 2^{b+1}$ **then**
  $\alpha_A \Leftarrow \frac{2^{b+1}}{|\textsc{Finger}([A-(\alpha_A-1),A+(\alpha_A-1)]|}$
end if

2.2.3 Network Size Estimate Error Bounds

Before we prove a bound on the error in the estimate for the network size and α, we include the definition of **healthy** which is discussed in greater detail in Section 2.6.

Definition 1. *The network is considered **healthy** if, for any two nodes A and B that correctly computed their respective estimates α_A and α_B, α_A and α_B do not differ by a factor greater than c.*

Theorem 1. *In a **healthy** network with N nodes, for any node A,*

$$\frac{N_A^{\frac{1}{2}}}{\frac{c^2}{2}N_A^{\frac{1}{2}} - c^2 + 2}N \leq N_A \leq \frac{c^2 N_A^{\frac{1}{2}}}{\frac{1}{2}N_A^{\frac{1}{2}} + 2c^2 - 1}N.$$

Proof. To prove the minimal and maximal estimates for the network size, we will construct

the two possible worst cases. A node A calculates an α_A estimate by satisfying the equation

$$\alpha_A = \frac{2^b}{N_A^{\frac{1}{2}}}.$$

Since the network is healthy, for any two nodes A and B, α_A and α_B can not differ by a factor greater than c. Hence

$$\alpha_A \leq c\alpha_B \qquad \text{and} \qquad \alpha_B \leq c\alpha_A.$$

We will construct the first worst case to be when a node A underestimates the size of the network. This occurs when the range $[A - \alpha_A, A + \alpha_A]$ contains the minimal number of nodes, while the remaining range $(A + \alpha_A, A - \alpha_A)$ has the maximal number of nodes. Let B be a node in the range $(A + \alpha_A, A - \alpha_A)$ with the greatest α estimate. Then we know that the following relationships hold by definition

$$\alpha_A = c\alpha_B \qquad \alpha_A = \frac{2^b}{N_A^{\frac{1}{2}}} \qquad \alpha_B = \frac{2^b}{N_B^{\frac{1}{2}}}$$

Thus the number of nodes for the segments $[A-\alpha_A, A+\alpha_A]$ and $[B-\alpha_B, B+\alpha_B]$ respectively are

$$2\frac{2^b}{\alpha_A} = 2N_A^{\frac{1}{2}} \qquad 2\frac{2^b}{\alpha_B} = 2N_B^{\frac{1}{2}}$$

Rearranging all equations in terms of N_A and b we obtain

$$\alpha_A = \frac{2^b}{N_A^{\frac{1}{2}}} \qquad \alpha_B = \frac{2^b}{cN_A^{\frac{1}{2}}} \qquad N_B^{\frac{1}{2}} = cN_A^{\frac{1}{2}} \tag{2.2}$$

Since the total number of nodes within the network can be computed as

$$(\#\text{ of } \alpha_A \text{ gaps}) \cdot (\#\text{ of nodes in } \alpha_A \text{ gap}) + (\#\text{ of } \alpha_B \text{ gaps}) \cdot (\#\text{ of nodes in } \alpha_B \text{ gap})$$

which is equivalent to

$$2 \cdot N_A^{\frac{1}{2}} + \frac{2^b - 2\alpha_A}{2\alpha_B} \cdot N_B^{\frac{1}{2}}.$$

By using the substitutions in (2.2) we obtain

$$\begin{aligned}
&2N_A^{\frac{1}{2}} + \frac{2^b - 2 \cdot \frac{2^b}{N_A^{\frac{1}{2}}}}{2 \cdot \frac{2^b}{cN_A^{\frac{1}{2}}}} \cdot cN_A^{\frac{1}{2}} \\
&2N_A^{\frac{1}{2}} + \frac{2^b N_A^{\frac{1}{2}} - 2^{b+1}}{N_A^{\frac{1}{2}}} \cdot \frac{cN_A^{\frac{1}{2}}}{2^{b+1}} \cdot cN_A^{\frac{1}{2}} \\
&2N_A^{\frac{1}{2}} + \left(\frac{N_A^{\frac{1}{2}}}{2} - 1\right) c^2 N_A^{\frac{1}{2}} \\
&\frac{c^2}{2} N_A - c^2 N_A^{\frac{1}{2}} + 2N_A^{\frac{1}{2}} \\
&N_A^{\frac{1}{2}} \left(\frac{c^2}{2} N_A^{\frac{1}{2}} - c^2 + 2\right)
\end{aligned}$$

Now A's estimate of the size of the network is N_A. Therefore, to calculate the error, we

compute $\frac{estimate}{actual}$

$$\frac{N_A}{N_A^{\frac{1}{2}}\left(\frac{c^2}{2}N_A^{\frac{1}{2}} - c^2 + 2\right)} \tag{2.3}$$

$$\frac{N_A^{\frac{1}{2}}}{\frac{c^2}{2}N_A^{\frac{1}{2}} - c^2 + 2}. \tag{2.4}$$

Finally, we will construct the second worst case to be when a node A overestimates the size of the network. This occurs when the range $[A - \alpha_A, A + \alpha_A]$ contains the maximal number of nodes, while the remaining range $(A + \alpha_A, A - \alpha_A)$ has the minimal number of nodes. Let B be a node in the range $(A + \alpha_A, A - \alpha_A)$ with the smallest α estimate. Then we know that the following relationships hold by definition

$$\alpha_A = \frac{\alpha_B}{c} \qquad \alpha_A = \frac{2^b}{N_A^{\frac{1}{2}}} \qquad \alpha_B = \frac{2^b}{N_B^{\frac{1}{2}}}$$

Hence the number of nodes for the segments $[A - \alpha_A, A + \alpha_A]$ and $[B - \alpha_B, B + \alpha_B]$ respectively are

$$2\frac{2^b}{\alpha_A} = 2N_A^{\frac{1}{2}} \qquad 2\frac{2^b}{\alpha_B} = 2N_B^{\frac{1}{2}}$$

Rearranging all equations in terms of N_A and b we obtain very similar equations as before

$$\alpha_A = \frac{2^b}{N_A^{\frac{1}{2}}} \qquad \alpha_B = \frac{2^b c}{N_A^{\frac{1}{2}}} \qquad N_B^{\frac{1}{2}} = \frac{N_A^{\frac{1}{2}}}{c} \tag{2.5}$$

Since the total number of nodes within the network can be computed as

$$(\#\text{ of } \alpha_A \text{ gaps}) \cdot (\#\text{ of nodes in } \alpha_A \text{ gap}) + (\#\text{ of } \alpha_B \text{ gaps}) \cdot (\#\text{ of nodes in } \alpha_B \text{ gap})$$

which is equivalent to

$$2 \cdot N_A^{\frac{1}{2}} + \frac{2^b - 2\alpha_A}{2\alpha_B} \cdot N_B^{\frac{1}{2}}.$$

By using the substitutions in (2.5) we obtain

$$\begin{gathered}
2N_A^{\frac{1}{2}} + \frac{2^b - 2 \cdot \frac{2^b}{N_A^{\frac{1}{2}}}}{2 \cdot \frac{2^b c}{N_A^{\frac{1}{2}}}} \cdot \frac{N_A^{\frac{1}{2}}}{c} \\
2N_A^{\frac{1}{2}} + \frac{2^b N_A^{\frac{1}{2}} - 2^{b+1}}{N_A^{\frac{1}{2}}} \cdot \frac{N_A^{\frac{1}{2}}}{c2^{b+1}} \cdot \frac{N_A^{\frac{1}{2}}}{c} \\
2N_A^{\frac{1}{2}} + (\frac{N_A^{\frac{1}{2}}}{2} - 1)\frac{N_A^{\frac{1}{2}}}{c^2} \\
\underline{\frac{\frac{1}{2}N_A + 2c^2 N_A^{\frac{1}{2}} - N_A^{\frac{1}{2}}}{c^2}} \\
\underline{\frac{N_A^{\frac{1}{2}}(\frac{1}{2}N_A^{\frac{1}{2}} + 2c^2 - 1)}{c^2}}
\end{gathered}$$

Now A's estimate of the size of the network is N_A. Therefore, to calculate the error, we

again compute $\frac{estimate}{actual}$ to obtain

$$\frac{N_A}{\frac{N_A^{\frac{1}{2}}}{c^2}\left(\frac{1}{2}N_A^{\frac{1}{2}}+2c^2-1\right)} \tag{2.6}$$

$$\frac{c^2N_A^{\frac{1}{2}}}{\frac{1}{2}N_A^{\frac{1}{2}}+2c^2-1} \tag{2.7}$$

Thus we have bounded from above and below the possible estimates for N_A. □

From the proof of Theorem 1 we can see that the dominant term in the bound is $N_A^{\frac{1}{2}}$. Thus, for large values of N_A, $\frac{1}{c^2}N \leq N_A \leq c^2N$ is a valid approximation. Intuitively, this bound makes sense because there are approximately $\alpha = N^{\frac{1}{2}}$ segments. Hence, all but one of these segments have an extra c or $\frac{1}{c}$ times as many nodes. Thus, in the two cases, we either under or over approximate by a factor of c^2. From this intuition, we can see it follows that

Corollary 1. *For a **healthy** network with N nodes, for any node A, then*

$$\mathcal{O}\left(\frac{\alpha}{c}\right) \leq \alpha_A \leq \mathcal{O}(c\alpha).$$

Proof. From Theorem 1 we know that

$$\frac{N_A^{\frac{1}{2}}}{\frac{c^2}{2}N_A^{\frac{1}{2}}-c^2+2}N \leq N_A \leq \frac{c^2N_A^{\frac{1}{2}}}{\frac{1}{2}N_A^{\frac{1}{2}}+2c^2-1}N.$$

Hence,

$$\frac{N_A^{\frac{1}{2}}}{\frac{c^2}{2}N_A^{\frac{1}{2}} - c^2 + 2}N = \mathcal{O}\left(\frac{N}{c^2}\right) \qquad \text{and} \qquad \frac{c^2 N_A^{\frac{1}{2}}}{\frac{1}{2}N_A^{\frac{1}{2}} + 2c^2 - 1}N = \mathcal{O}(c^2 N).$$

Since

$$N^{\frac{1}{2}} = \frac{2^b}{\alpha} \qquad \text{and} \qquad N_A^{\frac{1}{2}} = \frac{2^b}{\alpha_A}$$

then

$$\mathcal{O}\left(\frac{\alpha}{c}\right) \leq \alpha_A \leq \mathcal{O}(c\alpha).$$

□

2.2.4 Finger Table Size Bounds

To show that we satisfy the constraint of $\Theta(N^{\frac{1}{2}})$ nodes in the finger table, we start by proving bounds on the number of local peers. Then move on to prove bounds on the number of distant peers. By combining the two, we show that we satisfy the constraint.

Theorem 2. *In a **healthy** network with N nodes, the total number of **local peers** is bounded below by:*

$$\frac{2\sqrt{2N}}{c} - \frac{4}{c^2}$$

and bounded above by:

$$2c\sqrt{2N} + 4c^2$$

thus,

$$\Theta(N^{\frac{1}{2}}).$$

Proof. The number of local peers falls directly from Theorem 1. We know that a node A will have $\frac{2^b}{2\alpha_A} = 2N_A^{\frac{1}{2}}$ peers in the region $[A - \alpha_A, A + \alpha_A]$ around it. In the worst case, node A will have overestimated the size of the entire network to be $\frac{c^2 N_A^{\frac{1}{2}}}{\frac{1}{2}N_A^{\frac{1}{2}} + 2c^2 - 1}N$.

Hence,

$$N_A = \frac{c^2 N_A^{\frac{1}{2}}}{\frac{1}{2}N_A^{\frac{1}{2}} + 2c^2 - 1}N$$

$$N_A = \frac{2c^2 N_A^{\frac{1}{2}}}{N_A^{\frac{1}{2}} + 4c^2 - 2}N$$

$$N_A \cdot (N_A^{\frac{1}{2}} + 4c^2 - 2) = 2c^2 N_A^{\frac{1}{2}} N$$

$$N_A^{\frac{1}{2}} \cdot (N_A^{\frac{1}{2}} + 4c^2 - 2) = 2c^2 N \qquad \text{since } N_A > 0$$

$$N_A + (4c^2 - 2)N_A^{\frac{1}{2}} - 2c^2 N = 0$$

Solving for the quadratic roots we obtain

$$\frac{-4c^2 + 2 + (4 - 16c^2 + 16c^4 + 8c^2 N)^{\frac{1}{2}}}{2} \quad \text{and} \quad \frac{-4c^2 + 2 - (4 - 16c^2 + 16c^4 + 8c^2 N)^{\frac{1}{2}}}{2}.$$

Taking the positive root and since there are $2N_A^{\frac{1}{2}}$ peers in the region $[A - \alpha_A, A + \alpha_A]$, we

have at most

$$
\begin{aligned}
& 2N_A^{\frac{1}{2}} \\
&= -4c^2 + 2 + (4 - 16c^2 + 16c^4 + 8c^2N)^{\frac{1}{2}} \\
&\leq \sqrt{16c^4 + 8c^2N} \qquad \text{since } c \geq 1 \\
&\leq \sqrt{8c^2N} + \sqrt{16c^4} \\
&= 2c\sqrt{2N} + 4c^2
\end{aligned}
$$

local peers.

Similarly, in the worst case node A will have underestimated the size of the entire network to be $\frac{N_A^{\frac{1}{2}}}{\frac{c^2}{2}N_A^{\frac{1}{2}} - c^2 + 2}N$.

Hence,

$$
\begin{aligned}
N_A &= \frac{N_A^{\frac{1}{2}}}{\frac{c^2}{2}N_A^{\frac{1}{2}} - c^2 + 2}N \\
N_A^{\frac{1}{2}}\left(\frac{c^2}{2}N_A^{\frac{1}{2}} - c^2 + 2\right) &= N \qquad \text{since } N_A > 0 \\
\frac{c^2}{2}N_A + (2 - c^2)N_A^{\frac{1}{2}} - N &= 0
\end{aligned}
$$

Solving for the quadratic roots we obtain

$$
\frac{c^2 - 2 + (4 - 4c^2 + c^4 + 2c^2N)^{\frac{1}{2}}}{c^2} \qquad \text{and} \qquad \frac{c^2 - 2 - (4 - 4c^2 + c^4 + 2c^2N)^{\frac{1}{2}}}{c^2}.
$$

Taking the positive root and since there are $2N_A^{\frac{1}{2}}$ peers in the region $[A - \alpha_A, A + \alpha_A]$, we

have at least

$$
\begin{aligned}
& 2N_A^{\frac{1}{2}} \\
&= \frac{2c^2 - 4 + 2(4 - 4c^2 + c^4 + 2c^2N)^{\frac{1}{2}}}{c^2} \\
&\geq \frac{2\sqrt{2c^2N} + 2c^2 - 4}{c^2} \qquad \text{since } c \geq 1 \text{ and } c^4 - 4c^2 + 4 \geq 0 \\
&\geq \frac{2\sqrt{2N}}{c} - \frac{4}{c^2}
\end{aligned}
$$

local peers.

Since c is a constant, we have $\Theta(N^{\frac{1}{2}})$ local peers.

□

Theorem 3. *In a **healthy** network with N nodes, the total number of **distant peers** is bounded below by*

$$2\sqrt{N} - c - \frac{2}{c}$$

and bounded above by

$$c^2\sqrt{2N} + 2c^3$$

thus,

$$\Theta(N^{\frac{1}{2}}).$$

Proof. Let A be a node in the network. A will need at least one peer for each segment around the circle. Since the identifier space is of size 2^b, and each segment is $\frac{\alpha_A}{c}$, we can

derive that the number of distant peers is at most:

$$\frac{2^b - 2\alpha_A}{\frac{\alpha_A}{c}} = \frac{2^b c}{\alpha_A} - 2c$$

By definition we know that

$$\alpha_A = \frac{2^b}{N_A^{\frac{1}{2}}}.$$

Hence

$$\frac{2^b c}{\frac{2^b}{N_A}} - 2c = cN_A - 2c.$$

From Theorem 2, we can bound the size of N_A from above by

$$\begin{aligned}
& c\left(\frac{-4c^2 + 2 + (4 - 16c^2 + 16c^4 + 8c^2N)^{\frac{1}{2}}}{2}\right) - 2c \\
& \leq c\left(\frac{\sqrt{16c^4 + 8c^2N}}{2}\right) \qquad \text{since } c \geq 1 \\
& \leq c\left(\frac{\sqrt{8c^2N} + \sqrt{16c^4}}{2}\right) \\
& = c(c\sqrt{2N} + 2c^2) \\
& = c^2\sqrt{2N} + 2c^3
\end{aligned}$$

Similarly, we can bound the size of N_A from below by:

$$
\begin{aligned}
& c\left(\frac{c^2-2+(4-4c^2+c^4+2c^2N)^{\frac{1}{2}}}{c^2}\right)-2c \\
\geq & \left(\frac{c^2-2+\sqrt{2c^2N}}{c}\right)-2c \qquad \text{since } c \geq 1 \text{ and } c^4-4c^2+4 \geq 0 \\
\geq & 2\sqrt{N}-c-\frac{2}{c}
\end{aligned}
$$

Since c is a constant, we have $\Theta(N^{\frac{1}{2}})$ distant peers.

□

Theorem 4. *In a **healthy** network with N nodes, the total number of **peers** is:*

$$\Theta(N^{\frac{1}{2}})$$

Proof. This falls directly from the combination of Theorem 2 and Theorem 3. □

2.2.5 Simple Lookup

Before we continue on to how a node looks up a key k in constant time, we show a very simple (albeit poor) routing algorithm. We start by describing the algorithm, then move on to an example, and finally conclude with pseudocode.

Premise

The premise is that a node will do a linear search by traversing the edge of the ring. The edge of the ring is formed by left and right neighbour links. Thus, if node A is to look up

key k that it does not possess. Then without loss of generality, suppose the key k is to the left of A, then A will contact its left neighbour B. By continuing the traversal around the ring to the left, A will eventually find the node which is the successor of k.

Example

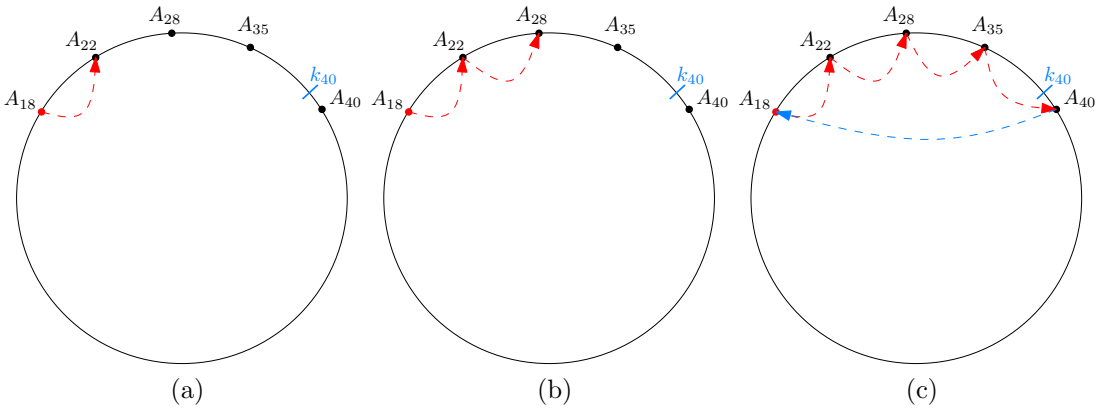


Figure 2.2: Simple lookup example

Suppose there existed node A_{18} that is to find the key k_{40}. A_{18} starts by noticing that k_{40} is to its right and decides to contact its right neighbour A_{22} as shown in Figure 2.2a. A_{22} notices that k_{40} is still to its right and decides to contact its right neighbour A_{28} as shown in Figure 2.2b. Finally, after a few more iterations A_{41} was contacted and it notices it is the successor of k_{40}. Thus, A_{41} finishes the lookup by responding to A_{18} as shown in Figure 2.2c.

Pseudocode

In the pseudocode of Algorithm 2, lines [7-8] check the exit condition for when we are proceeding to the right. While lines [9-10] check the exit condition for when we are

Algorithm 2 Simple lookup

```
 1: Preconditions:
 2: A is the current network node.
 3: k is the value of the key.
 4:
 5: LOOKUP(k)
 6: loop
 7:    if k ∈ (A, A.RIGHT] then
 8:       return A.RIGHT
 9:    else if k ∈ (A.LEFT, A] then
10:       return A
11:    else if A < k then
12:       A ⇐ A.RIGHT
13:    else
14:       A ⇐ A.LEFT
15:    end if
16: end loop
```

proceeding around the circle to the left. Finally, lines [11-12] and [13-14] respectively decide on whether we proceed to the right or left around the circle.

2.2.6 Constant Time Lookup

For a key to be found in 2 hops, we use the knowledge that any node will be able to find the successor in either its own or a distant peer's local peer set. Thus we move on to an example, and then prove that we can guarantee to find the proper successor in 2 hops. We conclude with some pseudocode describing the algorithm.

Example

Suppose there is a node A that wanted to lookup key k. A starts off by comparing whether k falls in its local peer set as shown in Figure 2.3a. A then looks at all of the distant peers

as shown in Figure 2.3b. Finally A selects the distant peer B which is closest to k as shown in Figure 2.3c. A asks B for the successor of k, which B then returns.

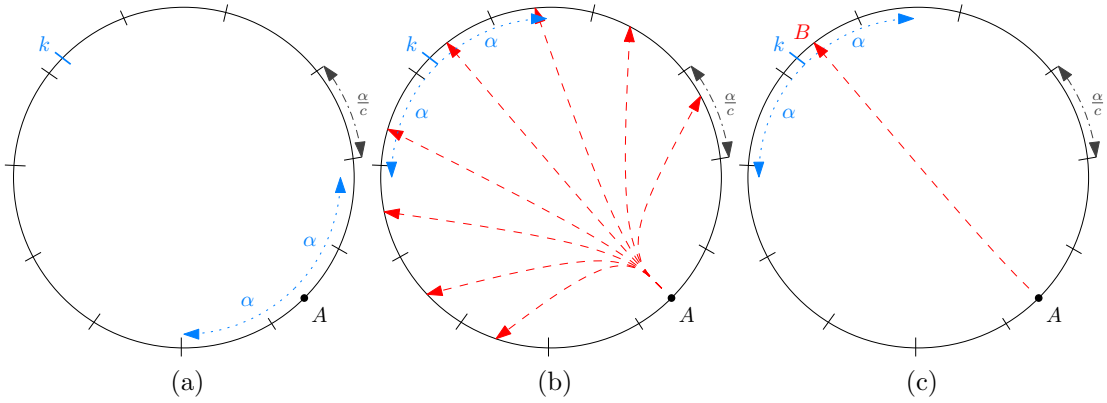


Figure 2.3: Fast lookup example

Proof

Theorem 5. *In a **healthy** network, as long as there were no node changes (joins/deaths) along the route, k can be found in 2 hops.*

Proof. Suppose node A wanted to lookup key k. We have two cases, either k is, or is not, in the range $[A - \alpha_A, A + \alpha_A]$. For the first case, suppose $k \in [A - \alpha_A, A + \alpha_A]$. Thus, by definition, $\text{SUCCESSOR}(k) \in A.\text{FINGER}$. And A will be able to query $\text{SUCCESSOR}(k)$, thus locating k in 1 hop.

For the second case, $k \notin [A - \alpha_A, A + \alpha_A]$, by definition

$$\exists B \in A.\text{FINGER} \text{ s.t. } B - k \leq \frac{\alpha_A}{c}.$$

Furthermore, by definition of a healthy network, we know that

$$\frac{\alpha_A}{c} \leq \alpha_B \leq c\alpha_A.$$

Thus, $k \in [B - \alpha_B, B + \alpha_B]$. Hence, SUCCESSOR($k$) $\in B$.FINGER and B will be able to return the address of SUCCESSOR(k) to A. Therefore A was able to lookup the SUCCESSOR(k) in 1 hop. Finally, A will be able to query SUCCESSOR(k), thus locating k in 2 hops. □

Pseudocode

Algorithm 3 Constant time lookup

```
1: Preconditions:
2: A is the current network node.
3: k is the value of the key.
4:
5: LOOKUP(k)
6: if A − k < α_A then
7:    return A.FINGER(k)
8: else
9:    B ⇐ A.FINGER(k)
10:   return B.SUCCESSOR(k)
11: end if
```

In the pseudocode of Algorithm 3, line 6 checks whether k is in the range $[A-\alpha_A, A+\alpha_A]$. If so, line 7 returns the SUCCESSOR(k) from the contents of A's finger table. Otherwise, line 9 finds the closest node to k within A's finger table which is guaranteed to be within $[A - \alpha_A, A + \alpha_A]$. While line 10 returns the SUCCESSOR(k) from the contents of B's finger table.

2.2.7 Lookups In An Unhealthy Network

Suppose we have an unhealthy network with no node changes (joins/deaths) along the routing path. Let A be a node where α_A is maximal in the network and let B be a node where α_B is minimal in the network. Since the maximal gap between distant peers in A's finger table is bounded by $\frac{2\alpha_A}{c}$, the maximal distance from any key k to a node within A's finger table is bounded by $\frac{\alpha_A}{c}$. Since each node in the network is responsible for a segment to its left and right of size at least α_B, then at most

$$\frac{\alpha_A}{c\alpha_B} = \frac{2^b}{cN_A^{\frac{1}{2}}} \cdot \frac{N_B^{\frac{1}{2}}}{2^b} = \frac{N_B^{\frac{1}{2}}}{cN_A^{\frac{1}{2}}}$$

hops towards k are required. Thus there are $\frac{N_B^{\frac{1}{2}}}{cN_A^{\frac{1}{2}}}$ hops required to lookup a value in an unhealthy network. This result is obvious from the fact that lookups are dependant on the difference between network size estimates and the parameter c "relaxes" the strict requirement on the knowledge of α and the size of the network N.

2.3 Joining

For a client to join any network, it must gain enough information to satisfy the requirements and constraints the network imposes. In this case, a client is required to:

1. Gather the network constants and create the identifier ring.
2. Split the identifier space into segments.
3. Choose an identifier.

4. Create its finger table and compute α.

5. Announce its join.

Initially, the client requires the knowledge of a node Z already in the network. The client contacts Z and obtains the constants b and c and Z's estimate α_Z. Thus the client now uses the value b to generate the identifier space $[0, 2^{b-1}]$.

With the identifier space created, the client is now required to split the identifier space into segments. To ensure that nodes boundaries between segments within the network remain randomly distributed, the client chooses a random integer v from within the identifier space as its first boundary. It then uses v and the segment size $\frac{\alpha_Z}{c}$ to partition the identifier space into $Y = \left\lceil \frac{2^b c}{\alpha_Z} \right\rceil$ such segments. The segment S_i is defined as the discrete range $(v + i\frac{\alpha_Z}{c}, v + (i+1)\frac{\alpha_Z}{c}]$ for $i \in \{0, \ldots, Y-1\}$.

For each segment, a node is contacted at random to gather information that the client will use to choose its own identifier. For each node R_i contacted, the information gathered contains the size of the largest gap G_i in their local peer set and its estimate α_{R_i}. Thus we now have three sets:

- the nodes $R = \{R_0, \ldots, R_{Y-1}\}$.
- the gaps $G = \{G_0, G_1, \ldots, G_{Y-1}\}$.
- the estimates $D = \{\alpha_0, \alpha_1, \ldots, \alpha_{Y-1}\}$.

The client uses the gaps and the estimates to choose an identifier for itself. The client begins by checking if the network is unhealthy by examining the estimates and noticing that $\textsc{Min}(D) < c \cdot \textsc{Max}(D)$.

If the network is unhealthy, then the client chooses the gap G_l corresponding to the largest α_l. If there is a tie, the tie is broken by choosing the segment with the largest gap. If the tie still persists, a gap is chosen at random from all those that are tied.

If the network is healthy, and there exists gaps which are at least twice the size of the smallest gap within G, then the client chooses the largest gap G_l from the set G. If there is a tie, the client chooses the segment with the largest α, if the tie still persists then the client chooses the gap at random from all those that are tied. If there does not exist such a G_l, then the client chooses the gap as if the network was unhealthy.

Using the G_l chosen, the client chooses its identifier to be $A = \textsc{MidPoint}(G_l)$. The client then looks up its own identifier acquiring $B = \textsc{Successor}(A)$, namely its right neighbour. Contacting B, the client acquires its left neighbour C. Thus the client now knows of its position within the identifier space and its two neighbours.

The client is now required to create its finger table and compute α. It turns out that the distant peers is precisely the set of nodes R. The client now must obtain its local peers, which it does so by asking B and C for all the local peers which are to the right and left of itself respectively. With all the local peers gathered, the client computes its estimate α_A. Note that if the network estimate α_A is less than the original value α_Z which was used to partition the identifier space, the client will have to look up nodes until it satisfies the constraint that the maximal size of a gap in its finger table is $\frac{\alpha_A}{c}$. This is discussed in further detail in Section 2.4.2.

Finally with the finger table constructed and α computed, the client announces that it has joined the network completing the join process. The discussion around A's join announcement is in Section 2.5.

Example

For a client wishing to join the network, it begins by contacting a bootstrap node Z from which it acquires the network parameters such as the constants b and c and also Z's estimate for α. The client then partitions the identifier space into segments of size $\frac{\alpha}{c}$ starting from a randomly chosen value v as illustrated in Figure 2.4a.

The client then begins to lookup a random node in each segment S_i and acquires from each R_i the size of the largest gap G_i in its local peer set and also its respective estimate α_{R_i}. The client uses all this information to chose an identifier A as shown in Figure 2.4b.

Finally, A creates its finger table by combining all distant peers with its left and right neighbour's peer sets. Then A announces its join to its left and right neighbours completing the join process as shown in Figure 2.4c.

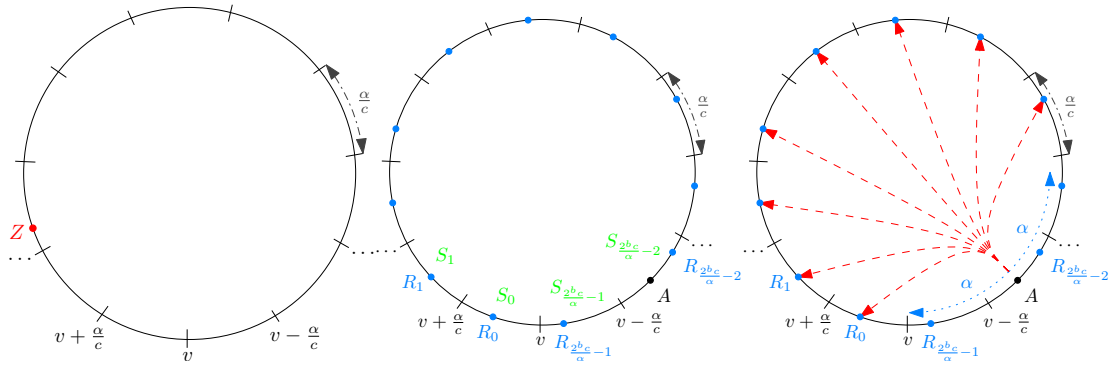


Figure 2.4: Join example

Pseudocode

In the pseudocode of Algorithm 4, line 5 begins with acquiring the network constants b and c. While line 6 generates the random value v which is used to distribute the boundaries

Algorithm 4 Joining the network

Preconditions:
Z is the initial bootstrap node.

$\textsc{Join}(Z)$
$\{b, c\} \Leftarrow Z.\{b, c\}$
$v \Leftarrow \textsc{Random}([0, 2^b))$
$A.\textsc{Finger} \Leftarrow \{Z\}$
$R = \{\}$
for $i = 0 \ldots \frac{\alpha_Z}{c}$ **do**
  $R \Leftarrow R \cup \{A.\textsc{Lookup}(\textsc{Random}((v + i \cdot \frac{\alpha_Z}{c}, v + (i+1) \cdot \frac{\alpha_Z}{c}]))\}$
end for
$G = \{\}$
$D = \{\}$
for all $R_i \in R$ **do**
  $G \Leftarrow G \cup R_i.\textsc{MaxGap}$
  $D \Leftarrow D \cup R_i.\alpha_{R_i}$
end for
$\{\alpha_{R_{j_0}}, \ldots, \alpha_{R_{j_p}}\} \Leftarrow \textsc{Min}(D)$
$\{\alpha_{R_{k_0}}, \ldots, \alpha_{R_{k_q}}\} \Leftarrow \textsc{Max}(D)$
if $\alpha_{R_{k_0}} > c\alpha_{R_{j_0}}$ **then**
  $A.\textsc{Id} \Leftarrow \textsc{Midpoint}(\textsc{Random}(\textsc{MaxGap}(\{G_{k_0}, \ldots, G_{k_q}\})))$
else
  $\{G_{m_0}, \ldots, G_{m_s}\} \Leftarrow \textsc{MinGap}(G)$
  $\{G_{d_0}, \ldots, G_{d_t}\} \Leftarrow \textsc{MaxGap}(G)$
  if $2G_{m_0} \leq G_{d_0}$ **then**
    $\{\alpha_{l_0}, \ldots, \alpha_{l_w}\} \Leftarrow \textsc{Max}(\{\alpha_{d_0}, \ldots, \alpha_{d_t}\})$
    $A.\textsc{Id} \Leftarrow \textsc{Midpoint}(\textsc{Random}(\{G_{l_0}, \ldots, G_{l_w}\}))$
  else
    $A.\textsc{Id} \Leftarrow \textsc{Midpoint}(\textsc{Random}(\textsc{MaxGap}(\{G_{k_0}, \ldots, G_{k_q}\})))$
  end if
end if
$B \Leftarrow R_y.\textsc{Predecessor}(A)$
$C \Leftarrow R_y.\textsc{Successor}(A)$
$A.\textsc{Finger} \Leftarrow B.\textsc{Finger}([A - \alpha_B, A)) \cup C.\textsc{Finger}((A, A + \alpha_C]) \cup R$
$\textsc{ComputeAlpha}()$
$A.\textsc{MaintainOnGrow}()$
$A.\textsc{Announce}(A, join)$

for segments amongst nodes uniformly at random. The client adds the bootstrap node to its finger table in line 7 which it then uses to lookup random nodes in each segment in lines [8-11]. Lines [12-17] gather the information for the maximal gap sizes and their α estimates. Lines [18-31] choose A's identifier, where [20-21] is if the network is detected to be unhealthy and lines [22-31] otherwise. Line 21 selects the midpoint of the largest gap for all the segments with the largest α's. If there is a tie, one of the largest gaps are selected at random. Lines [25-27] select the gap from the set of all gaps that are at least twice the size of the smallest gap. If no such gap exists, line 29 selects a gap as line 21 does. Finally, A gets its left and right neighbours B and C and creates its finger table on lines [32-34]. Line 35 computes α which is further discussed in Section 2.2.2. On line 36, A then verifies that the network constraints are still met by running the method MaintainOnGrow() which is further discussed in Section 2.4.2. Finally, line 37 concludes with A's join announcement which is discussed in Section 2.5.

2.4 Network Maintenance

An important issue of any network is the work required to maintain its structure and satisfy any constraints imposed. We begin this section by discussing the discovery of dead nodes. We then move on to discuss what action is required for a growing network. We conclude by covering what steps are required for a shrinking network.

2.4.1 Discovery Of Dead Nodes

A node may die gracefully, announcing its death to its neighbours, or may vanish inexplicably. For the latter case, we require a mechanism to discover failed nodes.

We employ a simple mechanism whereby if a message has not been received for some time from a nodes left or right neighbour, we periodically ping them. Therefore our detection mechanism is a simple keep alive scheme. Once a node is detected to have failed, the node is removed from the finger table and its death is announced. We discuss how the announcement is propagated in Section 2.5.

Pseudocode

Algorithm 5 Discovery of dead nodes

```
 1: Preconditions:
 2: A is the current network node.
 3:
 4: CheckDead()
 5: if ¬A.Ping(A.Left) then
 6:    dead ⇐ A.Left
 7:    A.Finger ⇐ A.Finger \ {dead}
 8:    A.Announce(dead, death)
 9: end if
10: if ¬A.Ping(A.Right) then
11:    dead ⇐ A.Right
12:    A.Finger ⇐ A.Finger \ {dead}
13:    A.Announce(dead, death)
14: end if
```

In the pseudocode of Algorithm 5, line 5 pings the left neighbour and if it has failed, lines [6-9] remove them from the finger table and announce its death. Similarly, lines [10-14] do the same action for the right neighbour.

2.4.2 A Growing Network

When the network is growing, the issue arises that the distance between adjacent distant peers in a finger table may become too large. This is the case when for a node A, A's finger table contains two adjacent nodes B and C such that $B < C$ and $B - C > \frac{2\alpha_A}{c}$. This is resolved by having node A choose a new random key r in the segment $\left(\frac{B+C}{2} - \frac{\alpha_A}{c}, \textsc{Min}\left(C.\textsc{Left}, \frac{B+C}{2} + \frac{\alpha_A}{c}\right)\right]$ and update A's finger table by adding the node $D = \textsc{Successor}(r)$. These steps are repeated for each such B and C in A's finger table until the constraint is satisfied. Note that if one allows for a possibility of failure, the constraint for the key r may be relaxed to the range $\left(\textsc{Max}\left(B, \frac{B+C}{2} - \frac{\alpha_A}{c}\right), \textsc{Min}\left(C, \frac{B+C}{2} + \frac{\alpha_A}{c}\right)\right)$ This reduces the cost of a lookup to find $C.\textsc{Left}$ at the possibility that $\textsc{Successor}(r) = C$, in which case a new value of r would have to be chosen until $\textsc{Successor}(r) \neq C$.

Pseudocode

Algorithm 6 Maintaining a growing network

```
1: Preconditions:
2: A is the current network node.
3:
4: MaintainOnGrow()
5: while ∃B, C ∈ A.Finger s.t. B + 2α_A/c < C, B and C are adjacent do
6:    D ⇐ A.lookup(Random((B+C)/2 − α_A/c, Min(C.Left, (B+C)/2 + α_A/c)]))
7:    A.Finger ⇐ A.Finger ∪ {D}
8: end while
```

In the pseudocode of Algorithm 6, line 5 finds all adjacent nodes B and C that do not satisfy the required constraint. Line 6 finds the successor of the random value in a range between B and C. Line 7 updates A's finger table.

2.4.3 A Shrinking Network

When the network is shrinking, we do not have to worry about adjacent nodes within a nodes finger table from being too distant. This is because the estimate α will be increasing since the amount of local peers is decreasing. Thus the constraint between adjacent distant peers will remain satisfied. We now have to worry that a node A may not know all its local peers. If we let α_A be the old segment size and α'_A be the new segment size then A is required to know all the peers to be able to answer any SUCCESSOR query in the region $[A-\alpha'_A, A+\alpha'_A]$. Thus, when the network is shrinking, a node A looks into its finger table for $B = \text{PREDECESSOR}(A-\alpha_A)$ and for $C = \text{SUCCESSOR}(A+\alpha_A)$. A then proceeds to ask for the left and right neighbours of B and C respectively until A is able to answer any lookup for the new region.

Pseudocode

Algorithm 7 Maintaining a shrinking network

```
1: Preconditions:
2: A is the current network node.
3:
4: MaintainOnShrink()
5: B ⇐ A.Successor(A − α_A)
6: while B ≥ A − α_A′ do
7:    B ⇐ B.Left
8:    A.Finger ⇐ A.Finger ∪ {B}
9: end while
10: C ⇐ A.Successor(A + α_A)
11: while C ≤ A + α_A′ do
12:    C ⇐ C.Right
13:    A.Finger ⇐ A.Finger ∪ {C}
14: end while
```

In the pseudocode of Algorithm 7, line 5 acquires the last known local peer to the furthest left of A. Lines [6-9] add all peers by walking along the perimeter of the circle to the left until the new bound is satisfied. Line 9 acquires the last known local peer to the furthest right of A. Lines [11-14] add all peers by walking along the perimeter of the circle to the right until the new bound is satisfied.

2.5 Network Messaging

Message passing in many distributed schemes either happens recursively, iteratively, or a combination of both. The problem with network messaging is what form of synchronization is required to correctly pass a message to all intended recipients. Since we have a very uniform set of messages, and we have a very specific set of recipients, we do not have to go to the detail as which would be required by a more complicated distributed system.

A simple method passing scheme is used. The nodes recursively pass the message around the ring starting from the originating node A in both the left and right directions up to a distance of α. This scheme has the advantage that the left and right links for a node are checked regularly as messages circulate around the ring.

Pseudocode

In the pseudocode of Algorithm 8, line 6 handles the message, whether it be a join or a death. Lines [7-9] announce the message to the left and right neighbours. Lines [10-13] handle messages which are traversing to the left, while lines [15-18] handle messages traversing to the right. Lines [11-13, 16-18] pass the message if it has not yet travelled far enough by using ones own estimate for α.

Algorithm 8 Simple message passing

```
1: Preconditions:
2: A is the current network node.
3: message is either join or death.
4:
5: ANNOUNCE(message)
6: A.HANDLEMESSAGE(message)
7: A.LEFT.ANNOUNCELEFT(message)
8: A.RIGHT.ANNOUNCERIGHT(message)
9:
10: ANNOUNCELEFT(message)
11: if A − B ≤ α_A then
12:     A.LEFT.ANNOUNCELEFT(message)
13: end if
14:
15: ANNOUNCERIGHT(mesage)
16: if A − B ≤ α_A then
17:     A.RIGHT.ANNOUNCERIGHT(message)
18: end if
```

2.6 Network Health

A network's health is a measure that is used to evaluate the quality of the network. When a network is considered to be healthy, then there is a guarantee that the size of the finger table will be $\mathcal{O}(N^{\frac{1}{2}})$ and that the lookup cost will be 2 hops. Thus it is important for any network to understand the actions which improve or deteriorate that health. We start by defining network health, we then discuss the actions which improve that health, and conclude with a discussion of the actions which weaken the network.

We repeat our definition of network health:

Definition 2. *The network is considered* ***healthy*** *if, for any two nodes A and B that correctly computed their respective estimates α_A and α_B, α_A and α_B do not differ by a factor greater than c.*

With a measure for health, then there is a sense that a network can be optimal. With that we can define network optimality:

Definition 3. *A network is considered* ***optimal****, if the distances between all adjacent nodes are equal.*

From the definition of optimality, we can see that it immediately follows that for any two nodes A and B, $\alpha_A = \alpha_B$. Also, when a network is optimal, the key distribution is uniform and any node may be chosen at random with high probability by just selecting the successor of a random integer in the identifier space. Thus, many beneficial properties arise from having an optimal network.

For our network, the only actions which change the distribution of nodes is when a node joins or departs. Generally we will see that when a node joins the network, it improves the health of the network, while when nodes depart we can approximate the amount of degradation.

Thus, we now argue that the network health generally improves when nodes are being added. In our join heuristic, we had two cases, the first case the network is unhealthy and the node is added to the segment with the largest α. Thus it is easy to see that the difference in the minimal and maximal α's around the circle has either stayed the same or decreased. The α will have only stayed the same if there were multiple segments with the same minimal α. There is a bound on such a number of segments, so eventually when enough nodes have been added, the distance between the minimal and maximal α's will have decreased.

The second case was when the network was healthy and the set of gaps were used to determine the join location. We can see that if there was a gap that was at least twice the

size of the smallest gap, then by splitting it in half we have created two equal size gaps These gaps are now each closer to the size of the smallest gap, thus the network has moved closer to optimality. If all the gaps are within a factor less than 2 of each other, then we use the same procedure as if the network was unhealthy to choose a gap and hence attempt to improve the network. Obviously, we begin to hamper the network when the network is close to or is optimal.

The other case which affects the networks health is when nodes leave the network. In this case, adjacent gaps are being merged and in general the size of the gaps is increasing. The problem is that network health is not measured by the difference in the sizes of the smallest and largest gaps. It is measured by the density of the segments around the circle. Thus we argue that if we assume each segment to be a bucket, and add a ball to each bucket representing a node death in said segment then this would be a close approximation. This is because as nodes die, α increases and in a sense one ball could be in two buckets at the same time. We will show through simulation the effect of removing nodes from the network would have on the density of segments around the circle. Now, if we begin with an optimal network, then we can state that there are $m = N^{\frac{1}{2}}$ segments. We assume that each segment has the same probability of being chosen, namely $\frac{1}{m}$. If $\frac{1}{k}$ of all the nodes die, then this follows a binomial distribution with $n = \frac{N}{k}$ and $p = \frac{1}{m}$. Hence the expected number of deaths per segment is

$$E[X] = np = \frac{N}{k}\frac{1}{N^{\frac{1}{2}}} = \frac{m}{k}.$$

and the variance in the number of deaths is

$$Var[X] = np(1-p) = \frac{N}{k} \cdot \frac{1}{m}\left(1 - \frac{1}{m}\right) = \frac{m^2}{mk} \cdot \left(\frac{m-1}{m}\right) = \frac{m-1}{k}.$$

We are interested in the distribution of the number of nodes in the segments. Hence, we look at the standard deviation

$$\sigma = Var[X]^{\frac{1}{2}} = \left(\frac{m-1}{k}\right)^{\frac{1}{2}} < \left(\frac{m}{k}\right)^{\frac{1}{2}}.$$

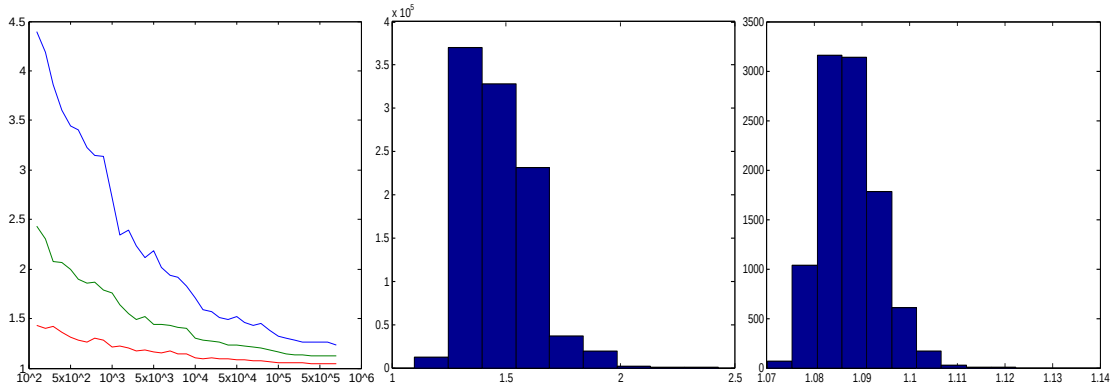
For a network to remain healthy, we would like to have no more than a factor of c between the minimal and maximal number of nodes in a bucket. Note that $\alpha = E[x] = \frac{m}{k}$.

$$\begin{aligned}
\left(\frac{m}{k}\right)^{\frac{1}{2}} &\leq \frac{c\alpha}{2} \\
\left(\frac{m}{k}\right)^{\frac{1}{2}} &\leq \frac{cm}{2k} \\
\frac{2k^{\frac{1}{2}}}{c} &\leq m^{\frac{1}{2}} \\
\frac{4k}{c^2} &\leq m
\end{aligned}$$

Thus, as a network increases in size, we can see that the stability of the network health also increases in view of large numbers of node deaths.

In Figure 2.5a we show the maximal ratio between the minimal and maximal alpha after 10000 rounds when 10%, 50%, and 90% of the nodes leave. In Figure 2.5b we show a histogram of the distribution for minimal and maximal alpha when half of a network containing 100 nodes leave over 1000000 trial runs. In Figure 2.5c we show a histogram of the distribution for minimal and maximal alpha when half of a network containing 1000000

nodes leave over 10000 trial runs.



(a) (b) (c)

Figure 2.5: Network health simulations

Chapter 3

RootChord Extensions

This chapter deals with ideas that do not affect the theoretical bounds of the network but have been shown to increase performance in practice. We discuss recommendations on how to replicate data, parallelize lookups, extend dead node discovery, implement remote procedure calls and pass messages in the network expediently.

3.1 Data Replication

The purpose of data replication is to increase the robustness of the network. If a node dies storing the only copy of data within the network, the data will have been lost. Thus many schemes for data replication have been studied such as using error correcting codes in Rainbow Skip Graphs [23] to add a level of robustness.

We recommend that data replication occur as it does in Chord [62], where up to r successors of a key k become the nodes that store replica copies of the data. This has

the advantage that these r successors naturally in the system already learn of the demise of neighbours, hence a suitable replacement can be found before all r replicas vanish. Secondly, if a node were to join, it would be able to assume the role of a replica and distribute the work of gathering all the data it must store from the previous r replicas. And, finally, having replicas allows one to relax the rate of convergence for join and death messages for local peers. This is because on a lookup, a node may return all the replicas which it knows of that store the data. Thus the system would only fail if all r replicas failed or were replaced by new nodes before these changes converged.

3.2 Parallelization Of Constant Time Lookup

The purpose of parallelization of lookups in a distributed hash table is to use the availability of multiple possible routes to the destination to our advantage. This advantage is two-fold, the first being that by using multiple routes, one can regularly get to the destination faster. The second, and more important advantage, is that we are able to avoid very slow or dead nodes. A study by Sairou *et al.* [61] on file sharing systems has shown that a significant fraction of nodes are connected over high latency / low bandwidth links. The presence of even one such slow logical hop on a path greatly increases the cost of the lookup. On the flipside, the problem with parallelization is obviously the added messaging cost and implementation complexity.

For the network protocol we have discussed, a node A is able to use a parallel lookup scheme when attempting to lookup a key k from a distant peer. It turns out that any distant peer in the range $[k - c\alpha_A, k + c\alpha_A]$ may be able to answer the query in a healthy network. Thus A uses a constant size subset (one including the peer A would have normally

chosen) of peers over the range $[k - c\alpha_A, k + c\alpha_A]$. This subset would contain at most x distant peers where x is a constant. x is considered to be the parallelization factor. We now proceed with an example followed by pseudocode.

Example

Let A be a node that is to look up key k as shown in Figure 3.1a. Assume that k would not be able to be answered from A's local peer set. Then A would look at the range between $[k - c\alpha, k + c\alpha]$ as shown in Figure 3.1b. A would then select all the distant peers falling into that range as the candidates used for the parallel lookup as shown in Figure 3.1c. The green dotted line represents the distant peer which A would have used for a regular lookup, while the red dashed lines represent peers which could be used to parallelize the lookup. A then chooses a constant size subset (one which includes the green dotted line distant peer) to use as parallel routes.

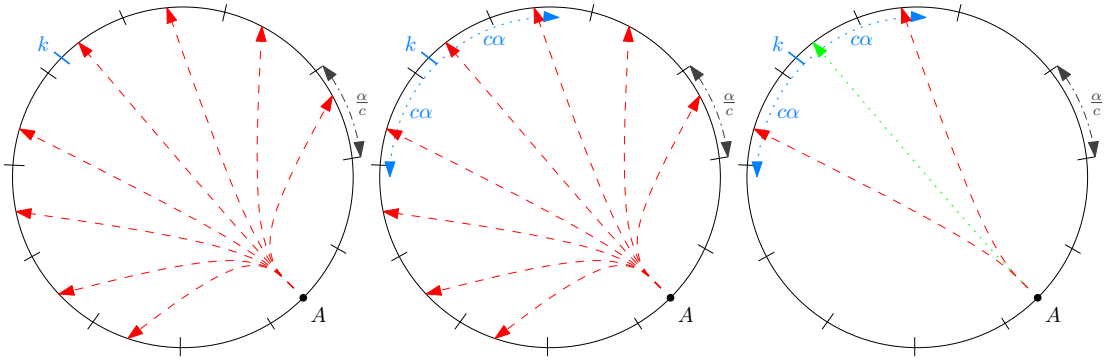


Figure 3.1: Parallel lookup example

Pseudocode

Algorithm 9 Parallel constant time lookup

```
1: Preconditions:
2: A is the current network node.
3: k is the key A wishes to lookup.
4: x is the parallelization factor (x ≥ 1).
5:
6: if k ∉ [A − α_A, A + α_A] then
7:    candidates ⇐ A.FINGER([k − cα_A, k + cα_A])
8:    candidates ⇐ (select x closest to k) ∈ candidates
9:    return candidates.FINGER(k)
10: else
11:    return A.FINGER(k)
12: end if
```

In the pseudocode of Algorithm 9, lines [6-9] handle the case for when a query can be parallelized. Line 7 selects all candidates from A's finger table. Line 8 reduces the candidates to a set of size x. Note that this set of candidates contains the distant peer A would have been used in a non-parallelized lookup. Line 9 calls a lookup on each of the candidate's finger table in parallel. Lines [10-11] handle the case when A does not need to contact a distant peer and is able to answer the lookup from its own finger table.

3.3 Extended Dead Node Discovery

A problem with our protocol is that there is no background dead node discovery for distant peers. For example, a node A will never remove a node outside of the range $[A-\alpha_A, A+\alpha_A]$ unless A attempts to contact it directly. A direct communication from A to all nodes in A's finger table outside of the range $[A - \alpha_A, A + \alpha_A]$ is unlikely. Thus a measure is required to prune dead links or it is highly likely that each node will accumulate many of them.

The idea is that whenever a node A communicates with a distant peer B, A will query B about the status of nearby distant peers to attempt to reduce the number of dead links.

Each time A communicates with B, A piggy backs on its original request to B a list of all the distant peers A knows of in the range $[B - c\alpha_A, B + c\alpha_A]$. Thus when B responds to the original request from A, it also returns the status of the nodes A queried about. The status for each node returned can either be that it *exists*, is *dead*, or *unknown*. For each node that is *dead*, B also responds with a suitable replacement, namely the closest node to the *dead* node. Since B has knowledge of the nodes in $[B - \alpha_B, B + \alpha_B]$, it will be able to answer about the status of nodes over that range. The problem lies when these two intervals differ and for any nodes that fall outside of B's range, it must respond *unknown*.

Pseudocode

Algorithm 10 Extended dead node discovery

```
1: Preconditions:
2: A is the current network node.
3: B is the network node A is about to communicate with.
4: request is the original request.
5:
6: candidates ⇐ A.FINGER([B − cα, B + cα]) \ A.FINGER([A − α, A + α])
7: {result, candidateInfo} ⇐ B.MESSAGE(request, candidates)
8: for all dead, replacement ∈ candidateInfo do
9:    A.FINGER ⇐ A.FINGER \ {dead} ∪ {replacement}
10: end for
```

In the pseudocode of Algorithm 10, line 6 gathers the candidates from A's finger table. Line 7 communicates with B and stores the results and information about the candidates. Lines [8-10] replace all dead nodes with their replacements.

3.4 Remote Procedure Calls

A remote procedure call is a protocol that one application uses to request a service from an application located remotely. The difference between a local and remote call to the initiating application is made transparent by the underlying framework providing the service. It may be that the remote procedure call is overlaid on top of the operating system or another framework. Remote procedure calls were initially created so that an application may be able to communicate with other components without a direct understanding of the underlying network, whether this be the Internet or a computing cluster. Generally, remote procedure calls are of three forms when recursion is involved. The first is purely recursive, the second is semi-recursive, and finally the last one is purely iterative. We will discuss all three, but first form the basis from which the examples in the sections below are derived. Figure 3.2 contains nodes A, B, C and D. The green node A will be the originator of the call while the blue node D will be the desired destination of the call. Finally, A knows only of B, B knows only of C, and C knows only of D. Let black solid lines represent calls while red dashed lines represent returns.

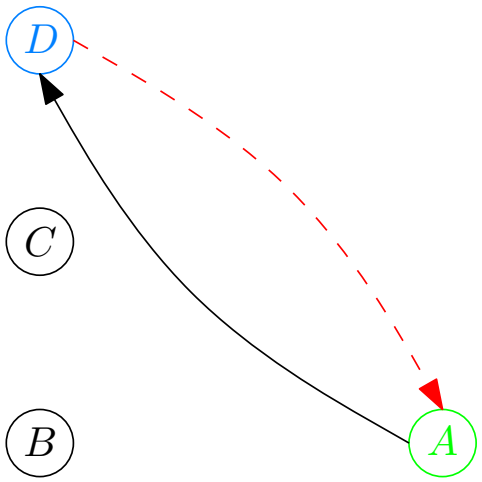


Figure 3.2: A remote procedure call

Recursive Remote Procedure Calls

A recursive remote procedure call occurs when A contacts B, and B contacts C passing on A's request. Furthermore, C will pass on A's request to the destination D. The response from D returns to C, then to B, and finally to A.

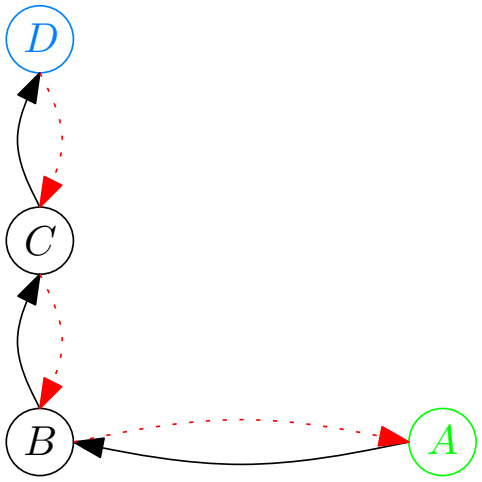


Figure 3.3: A recursive remote procedure call

The advantage of this approach is two-fold, the first being that the underlying network only needs be symmetric. Thus it works in schemes such as IPv4 where NAT may occur. Secondly, the contact information for D is never known to A, thus a level of secrecy is maintained. The major disadvantage of such a scheme is that the parameters of the procedure and return value must be passed to all intermediary nodes. This could be very expensive and is dependant on the size of the parameters and return value.

Semi-Recursive Remote Procedure Calls

The second type of remote procedure call is semi-recursive (*i.e.* essentially tail recursion [30]). Reusing our example from the purely recursive remote procedure call, the primary difference is that not only A's parameters are passed onto C from B, but also A's identity and contact information as the originating caller. Then this same information is

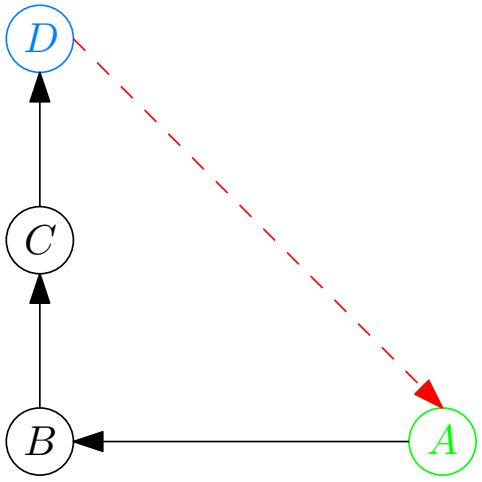


Figure 3.4: A semi-recursive remote procedure call

passed onto D. Where D, instead of returning the response to C, it closes the loop and delivers the response directly to A.

The advantage of such a scheme is that in a homogenous network where the communication times between nodes are approximately the same, this becomes the fastest scheme to make procedure calls when the size of the parameters of the procedure calls are small. The disadvantage of such a scheme is that that the parameters must still be passed to all intermediary nodes. As in recursive procedure calls, this could be very expensive and is solely dependant on the size of the parameters.

Iterative Remote Procedure Calls

The final, and most widely used, method is an iterative approach. The difference between the recursive approaches and the iterative approach is that if A wants to contact D, then it asks B for D's address. B replies with C's address as it believes that C is closer to D. Finally, A contacts C and acquires D's address and thus A contacts D directly.

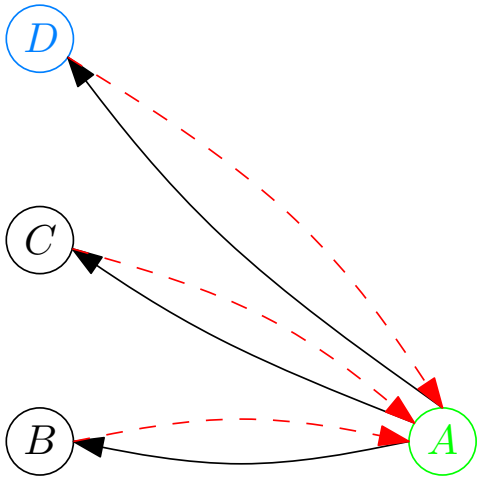


Figure 3.5: A recursive remote procedure call

Again the advantage of this approach is two-fold. The first being that if the parameters and return values from a procedure are large in comparison to the underlying communication about D's contact information, then it is only transferred once. The second advantage is that the underlying framework is able to cache C and D's address, and any further calls to C or D will not

require contacting B. The disadvantage of this methodology is that in practice this is the slowest method if D is only contacted once. This occurs because many underlying frameworks make links between nodes that can communicate with each other quickly. Thus the distance travelled for all the messages is usually greater than the other two approaches.

Conclusion

If the protocol is used to store vast quantities of data then the purely recursive approach is not suitable because of the high communication cost of retrieving information. Similarly, for data replication of large volumes of storage, either a high price is paid to pass on the data to be stored to distant nodes, or a limit is placed on which nodes the data can be replicated onto. Thus the iterative approach is most suited for the protocol because the high availability of node contact information allows for greater caching of information. It also allows for the greatest flexibility for data retrieval and replication.

3.5 Fast Network Messaging

By only passing messages on the outside of the ring, the rate at which the network converges is $\Theta(N^{\frac{1}{2}})$ (*i.e.* if a node drops out, $\Theta(N^{\frac{1}{2}})$ messages are sent sequentially to update the network). Thus to increase the rate at which the system converges, messages need to be passed to more than just the left and right neighbours but care has to be taken to not miss any nodes or duplicate work.

The issue is that there is a possibility that the node that the message originated from may yet not know of a node that had just recently joined. This is resolved by having each

node which receives a message to check whether there are any nodes within a range given to it and further pass the message a long. Below we provide an algorithm which places the burden of delivery on the originating node hence making the system converge at $\mathcal{O}(1)$ rate. There are many ways which we could adapt this to make it a recursive process making the system converge at a logarithmic rate such as in D1HT [51] and shown in Figure 1.2.

Pseudocode

In the pseudocode of Algorithm 11, lines [6-12] announce the messages to all A's local peers handing them a section which they are responsible for. Lines [15, 23] handle whether the message is a join or death. Lines [16-20, 24-28] handle passing the message on to any nodes that may have recently joined.

Algorithm 11 Fast message passing

```
1: Preconditions:
2: A is the current network node.
3: B is the node the message is about.
4: message is either join or death.
5:
6: Announce(B, message)
7: for all a_i ∈ A.Finger(A − α, A) do
8:    a_i.AnnounceLeft(B, message, (A.Predecessor(a_i), a_i))
9: end for
10: for all a_i ∈ A.Finger(A, A + α) do
11:    a_i.AnnounceRight(B, message, (a_i, A.Successor(a_i)))
12: end for
13:
14: AnnounceLeft(B, mesage, range)
15: A.HandleMessage(message)
16: if A − B ≤ α then
17:    for all a_i ∈ A.Finger(range) do
18:       a_i.AnnounceLeft(B, message, (A.Predecessor(a_i), a_i))
19:    end for
20: end if
21:
22: AnnounceRight(B, mesage, range)
23: A.HandleMessage(message)
24: if A − B ≤ α then
25:    for all a_i ∈ A.Finger(range) do
26:       a_i.AnnounceRight(B, message, (a_i, A.Successor(a_i)))
27:    end for
28: end if
```

Chapter 4

Conclusion

We have introduced "RootChord", a new distributed hash table that is able to dynamically adapt to the size of the network. This protocol is fault tolerant, parallelizable, and is able to do queries within 2 hops at the cost of storing a $\Theta(N^{\frac{1}{2}})$ size routing table in each node. The protocol considers all nodes to be equal, and hence no node is burdened with additional work. Nodes within the network are able to accurately estimate the size of the network. These network size estimates are off by no more than a factor of approximately c^2 of the actual network size.

Additionally, we have presented a detailed description and pseudocode for all major aspects of the protocol including that of lookups, joins, deaths, network maintenance and messaging. We have extended lookups to be intelligently parallelizable and have shown the cost of a lookup when the network becomes "unhealthy". We have explored dead node discovery and given a technique which is "lazy". Lazy in the sense that it will only acquire about nodes which it is interested in and ignore all other nodes. We have extended this

technique with a sense of locality, thus nodes which are near what is queried will also be checked. We have also considered some of the practical implications to the design as how to message, replicate data, and even which type of remote procedure call most benefits this DHT. We have explored the issue of network convergence with respect to messaging and showed multiple messaging protocols which would be suitable for different deployment scenarios.

There are both practical and theoretical questions that are left remaining. In practice, we need to see how this protocol actually handles being in the wild. Whether it is truly able to adapt to many network environments and what is the actual cost of storing such a large routing table compared to other DHTs. An analysis of the number of stale records over time would also be beneficial and finally what occurs to the distribution of identifiers under our join protocol and how far away it is from optimal in practice.

On the theoretical side, a better approximation to the effect node deaths have on the network is required. Future research could address the questions of whether this ring structure where neighbours almost form "cliques" have other applications beyond this DHT. What properties do these not so perfect "cliques" exhibit? Can we design a structure from these "cliques" which is able to route in $\mathcal{O}(d)$ yet only have a finger table of size $\mathcal{O}(N^{\frac{1}{d}})$?

References

[1] Azureus. http://www.vuze.com/. 3

[2] BGP. http://www.cisco.com/en/US/docs/internetworking/technology/handbook/bgp.html.

[3] Gnutella. http://www.gnutellaforums.com/. 1

[4] IGRP. http://www.cisco.com/en/US/tech/tk365/technologies_white_paper09186a00800c8ae1.shtml. 7

[5] Kazaa. http://www.kazaa.com. 1

[6] Napster. http://www.napster.com.

[7] OSPF. http://www.cisco.com/en/US/tech/tk365/technologies_white_paper09186a0080094e9e.shtml. 7

[8] PlanetLab. http://www.planet-lab.org. 3

[9] *1st Symposium on Networked Systems Design and Implementation (NSDI 2004), March 29-31, 2004, San Francisco, California, USA, Proceedings*. USENIX, 2004. 62

[10] André Allavena, Alan Demers, and John E. Hopcroft. Correctness of a gossip based membership protocol. In *PODC '05: Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 292–301, New York, NY, USA, 2005. ACM. 7

[11] Lars Arge, David Eppstein, and Michael T. Goodrich. Skip-webs: efficient distributed data structures for multi-dimensional data sets. In *PODC '05: Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 69–76, New York, NY, USA, 2005. ACM.

[12] James Aspnes and Gauri Shah. Skip graphs. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 384–393, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[13] N.T.J. Bailey. *Epidemic Theory of Infectious Diseases and its Applications*. Hafner Press, second edition edition, 1975. 7

[14] Philip A. Bernstein and Nathan Goodman. Concurrency control in distributed database systems. *ACM Comput. Surv.*, 13(2):185–221, 1981.

[15] Queries Ashwin Bharambe and Ashwin R. Bharambe. Mercury: Supporting scalable multi-attribute range. In *In SIGCOMM*, pages 353–366, 2004. 4

[16] K. Mani Chandy and Leslie Lamport. Distributed snapshots: determining global states of distributed systems. *ACM Trans. Comput. Syst.*, 3(1):63–75, 1985.

[17] Frank Dabek, Jinyang Li, Emil Sit, James Robertson, M. Frans Kaashoek, and Robert Morris. Designing a dht for low latency and high throughput. In *NSDI* [9], pages 85–98.

[18] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *PODC '87: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12, New York, NY, USA, 1987. ACM. 7

[19] Aisling O' Driscoll, Susan Rea, and Dirk Pesch. Performance evaluation and modeling of the chord dht structured overlay for ad-hoc networks, 2008.

[20] Jarret Falkner, Michael Piatek, John P. John, Arvind Krishnamurthy, and Thomas Anderson. Profiling a million user dht. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 129–134, New York, NY, USA, 2007. ACM. 3

[21] Pierre Fraigniaud and Philippe Gauron. D2b: A de bruijn based content-addressable network. *Theoretical Computer Science*, 355(1):65–79, 2006. Complex Networks.

[22] L. Garces-Erice, K. W. Ross, E.W. Biersack, P. A. Felber, and G. Urvoy-Keller. Topology-centric look-up service. In *in COST264/ACM Fifth International Workshop on Networked Group Communications (NGC*, pages 58–69. Springer, 2003.

[23] Michael T. Goodrich, Michael J. Nelson, and Jonathan Z. Sun. The rainbow skip graph: a fault-tolerant constant-degree distributed data structure. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 384–393, New York, NY, USA, 2006. ACM. 3, 48

[24] Rachid Guerraoui, Sidath B. Handurukande, Kevin Huguenin, Anne-Marie Kermarrec, Fabrice Le Fessant, and Etienne Riviere. Gosskip, an efficient, fault-tolerant and self organizing overlay using gossip-based construction and skip-lists principles. In *P2P '06: Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing*, pages 12–22, Washington, DC, USA, 2006. IEEE Computer Society. 4

[25] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of dht routing geometry on resilience and proximity. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 381–394, New York, NY, USA, 2003. ACM.

[26] Anjali Gupta, Barbara Liskov, and Rodrigo Rodrigues. Efficient routing for peer-to-peer overlays. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 9–9, Berkeley, CA, USA, 2004. USENIX Association. 4, 8

[27] Indranil Gupta, Ken Birman, Prakash Linga, Al Demers, and Robbert van Renesse. Kelips: Building an efficient and stable p2p dht through increased memory and background overhead. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03*, 2003. 4, 8

[28] Indranil Gupta, Anne-Marie Kermarrec, and Ayalvadi J. Ganesh. Efficient and adaptive epidemic-style protocols for reliable and scalable multicast. *IEEE Trans. Parallel Distrib. Syst.*, 17(7):593–605, 2006.

[29] Nicholas J.A. Harvey, A Nicholas J., Harvey John, John Dunagan, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. Skipnet: A scalable overlay network with practical locality properties, 2003. 4

[30] Jani Hautakorpi and Gonzalo Camarillo. Evaluation of dhts from the viewpoint of interpersonal communications. In *MUM '07: Proceedings of the 6th international conference on Mobile and ubiquitous multimedia*, pages 74–83, New York, NY, USA, 2007. ACM. 54

[31] David R. Jefferson. Virtual time. *ACM Trans. Program. Lang. Syst.*, 7(3):404–425, 1985.

[32] Norman L. Johnson and Samuel Kotz. *Urn Models and Their Application: An Approach to Modern Discrete Probability Theory (Probability & Mathematical Statistics).* John Wiley & Sons Inc, 1977.

[33] Frans Kaashoek and David R. Karger. Koorde: A simple degree-optimal distributed hash table. pages 98–107, 2003.

[34] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663, New York, NY, USA, 1997. ACM. 8

[35] Daishi Kato and Toshiyuki Kamiya. Evaluating dht implementations in complex environments by network emulator. In *IPTPS 07: Proceedings of the 6th International Workshop on Peer-to-Peer Systems*, February 2007.

[36] David Kempe, Jon Kleinberg, and Alan Demers. Spatial gossip and resource location protocols. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 163–172, New York, NY, USA, 2001. ACM. 7, 8

[37] Jon Kleinberg. The small-world phenomenon: an algorithm perspective. In *STOC '00: Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 163–170, New York, NY, USA, 2000. ACM. 6

[38] Ben Leong, Barbara Liskov, and Erik D. Demaine. Epichord: Parallelizing the chord lookup algorithm with reactive routing state management. In *IN PROCEEDINGS OF THE 12TH INTERNATIONAL CONFERENCE ON NETWORKS*, pages 1243–1259, 2004.

[39] Daniel Lewin. Consistent hashing and random trees : algorithms for caching in distributed networks. Master's thesis, Massachusetts Institute of Technology, 1998. 8

[40] Jinyang Li, Jeremy Stribling, Thomer M. Gil, Robert Morris, and M. Frans Kaashoek. Comparing the performance of distributed hash tables under churn. In *Proc. of the 3rd International Workshop on Peer-to-Peer Systems*, February 2004.

[41] Jinyang Li, Jeremy Stribling, Robert Morris, and M. Frans Kaashoek. Bandwidth-efficient management of dht routing tables. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 99–114, Berkeley, CA, USA, 2005. USENIX Association. 4

[42] Jinyang Li, Jeremy Stribling, Robert Morris, M. Frans Kaashoek, and Thomer M. Gil. A performance vs. cost framework for evaluating dht design tradeoffs under churn. *IEEE INFOCOM*, 1:225–236, 2005.

[43] David Liben-Nowell, Hari Balakrishnan, and David Karger. Analysis of the evolution of peer-to-peer systems. In *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 233–242, New York, NY, USA, 2002. ACM.

[44] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. In *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 183–192, New York, NY, USA, 2002. ACM. 4

[45] Gurmeet Singh Manku, Mayank Bawa, Prabhakar Raghavan, and Verity Inc. Symphony: Distributed hashing in a small world. In *In Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, pages 127–140, 2003. 4

[46] Gurmeet Singh Manku, Moni Naor, and Udi Wieder. Know thy neighbor's neighbor: the power of lookahead in randomized p2p networks. In *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 54–63, New York, NY, USA, 2004. ACM.

[47] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 53–65, London, UK, 2002. Springer-Verlag. 3

[48] David L. Mills. Network time protocol (ntp). Technical report, RFC, United States, 1985.

[49] David L. Mills. On the accuracy and stablility of clocks synchronized by the network time protocol in the internet system. *SIGCOMM Comput. Commun. Rev.*, 20(1):65–75, 1990.

[50] David L. Mills. *Computer Network Time Synchronization: The Network Time Protocol*. CRC Press, Inc., Boca Raton, FL, USA, 2006.

[51] Luiz Monnerat and Luiz R. Monnerat. D1ht: A distributed one hop hash table. Technical report, In Proc of the 20th IEEE Intl Parallel & Distributed Processing Symposium (IPDPS, 2005. 4, 57

[52] J. Ian Munro, editor. *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*. SIAM, 2004. 66

[53] S. Naicken, B. Livingston, A. Basu, S. Rodhetbhai, I. Wakeman, and D. Chalmers. The state of peer-to-peer simulators and simulations. *SIGCOMM Comput. Commun. Rev.*, 37(2):95–98, 2007.

[54] Moni Naor, Udi Wieder, and Small Worlds. Know thy neighbor's neighbor: Better routing for skip-graphs and small worlds, 2004.

[55] C. Greg Plaxton, Rajmohan Rajaraman, Andrea W. Richa, and Andr'ea W. Richa. Accessing nearby copies of replicated objects in a distributed environment. pages 311–320, 1997. 1

[56] Dongyu Qiu and R. Srikant. Modeling and performance analysis of bittorrent-like peer-to-peer networks. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 367–378, New York, NY, USA, 2004. ACM.

[57] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM. 2

[58] Sean Rhea, Byung-Gon Chun, John Kubiatowicz, and Scott Shenker. Fixing the embarrassing slowness of opendht on planetlab. In *WORLDS'05: Proceedings of the 2nd conference on Real, Large Distributed Systems*, pages 25–30, Berkeley, CA, USA, 2005. USENIX Association.

[59] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling churn in a dht. In *ATEC '04: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association. 3

[60] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001. Springer-Verlag. 2

[61] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A measurement study of peer-to-peer file sharing systems. 2002. 49

[62] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, 2003. 1, 2, 5, 10, 11, 48

[63] Kevin C. Zatloukal and Nicholas J. A. Harvey. Family trees: an ordered dictionary with optimal congestion, locality, degree, and search time. In Munro [52], pages 308–317.

[64] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and. Technical report, Berkeley, CA, USA, 2001. 2