DIPSEA: A MODULAR DISTRIBUTED HASH TABLE

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Gurmeet Singh Manku

September 2004

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Rajeev Motwani
(Department of Computer Science, Stanford University)
(Principal Adviser)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Hector Garcia-Molina
(Department of Computer Science, Stanford University)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Hari Balakrishnan
(Department of Electrical Engineering and Computer Science,
Massachusetts Institute of Technology)

Approved for the University Committee on Graduate Studies.

Abstract

A Distributed Hash Table (DHT) is a giant hash table that is cooperatively maintained by a large number of machines worldwide. The machines join and leave the system autonomously. The unprecedented scale and dynamism of the system calls for novel design techniques which emphasize decentralization and automatic re-configuration. Briefly, each machine in a DHT is assigned an ID in $\mathcal{I} = [0, 1)$. The set of IDs divides $\mathcal{I}$ into disjoint partitions, managed by one machine each. As a function of their IDs, the machines set up connections among themselves. These connections are used for routing messages between the machines. The challenge lies in devising efficient *decentralized* algorithms for ID management and connection maintenance.

We propose Dipsea, a modular architecture for building DHTs, consisting of three layers: ID Management, Overlay Routing and Data Management – this thesis focuses on the first two layers. The modularity of Dipsea imbues the overall system with several good properties. A large complex problem is broken down into smaller sub-problems, each of which can be attacked more or less independently. This contributes to reduction in complexity — it is possible to explore the design space of a sub-problem without being encumbered by its interactions with other sub-problems. Furthermore, the best solutions for individual sub-problems can be identified and put together to arrive at an overall design that is far more powerful than a design arrived at by a holistic approach.

Dipsea places existing DHT designs and improvements suggested for various DHTs into a common algorithmic framework. A significant accomplishment is the identification of layers and modules which are cleanly separated on the basis of functionality. Then for each module, we devise and analyze efficient algorithms – almost all of the algorithms we propose are the currently best-known algorithms for the corresponding modules.

Highlights of our contributions in ID Management and Overlay Routing layers:

1. A simple, decentralized ID Management algorithm which is independent of the Overlay Routing layer. The algorithm requires $O(R + \log n)$ messages and only one re-assignment of existing IDs in response to arrival or departure of machines. R denotes the average number of messages required by the Overlay Routing layer, and n denotes the current number of machines in the system.

2. A generalization of the above scheme whose analysis requires the solution to a novel Structured Coupon Collection Problem over cliques with multiple-choices per trial.

3. The design of an Emulation Engine which can emulate arbitrary families of deterministic and randomized routing networks. The Emulation Engine makes the design of Dipsea a significant improvement over existing DHT implementations, all of which are tied to specific families of routing networks.

4. Characterization of shortest paths in Chord, a family of deterministic routing networks that has been designed for DHTs.

5. The design of Papillon, a butterfly-based family of graphs defined over nodes placed in a circle. Papillon supports efficient greedy routing, in which each node forwards a message along that out-going edge which reduces the clockwise distance to the destination by the largest amount. In an n-node graph, Papillon routes in $O(\log n / \log d)$ hops with d links per node, which is asymptotically optimal.

6. The design of Symphony, one of the first randomized routing networks proposed for DHT routing. With k links per node, clockwise greedy routing takes $O(\frac{1}{k} \log^2 n)$ hops on average.

7. Tight analysis of clockwise greedy routing with/without lookahead in several randomized routing networks including Symphony, randomized Chord, and randomized hypercube. The idea underlying "greedy with lookahead" is to allow a node to use knowledge of its neighbor's neighbors for better routing decisions. We show that greedy routing without lookahead requires $\Theta(\log n)$ hops on average whereas greedy with lookahead entails only $\Theta(\log n / \log \log n)$ hops on average.

8. The design of Mariposa, an interesting combination of butterfly networks and Kleinberg's small-world construction. Mariposa also offers routes of $O(\log n / \log d)$ hops in the worst case, with d out-going links per node.

Acknowledgements

Several persons contributed to bring this thesis to fruition. I thank my adviser, Rajeev Motwani, for the guidance and the freedom he gave to me. Without his help, this thesis would not have materialized. Thanks to Prabhakar Raghavan for being my mentor for several projects. Thanks to my Reading Committee members: Hari Balakrishnan, Hector Garcia-Molina and Rajeev Motwani. I also thank Ashish Goel, Balaji Prabhakar and Scott Shenker for being on my Orals Committee despite their busy schedules.

Special thanks to my wife, Uma, without whose patience and perseverance, I would not have managed to vanish for days chasing paper deadlines. Many thanks to my parents for their blessings, and inculcating the love for knowledge in me.

Finally, I thank my colleagues at Stanford with whom I had innumerable discussions, and who gave a lot of feedback on drafts of my papers, my practice talks and my resume: Arvind, Shivnath, Mayank, Ramesh, Prasanna, Krishnaram, Shankar, Dilys, ... Special thanks to Mayank Bawa with whom I started brain-storming in mid-2000 for P2P-related ideas. It was the Symphony paper with Mayank that triggered a series of papers on Distributed Hash Tables which constitute this thesis. Finally, many thanks to members of the Systems-Quals study-group members from whom I learned a lot: Shivnath, George, Ramesh, Neil, Daniel, T. J., Emre, Sergio, Costas and Ed.

Contents

List of Tables

List of Figures

Chapter 1

Dipsea: An Overview

Defining interfaces is the most important part of system design.

BUTLER LAMPSON [L83]

A Distributed Hash Table (DHT) is a giant hash table that is maintained by a large number of machines spread across the world. The hash table is split into disjoint *partitions*. Each machine is assigned ownership of one partition, thereby making it the *manager* of that partition. The set of machines participating in the hash table is both dynamic and large-sized. This makes *decentralization* and *automatic re-configuration* two important design goals. The emphasis on decentralization stems from concerns of scalability. Automatic re-configuration is motivated by ease of management.

A DHT can serve as a repository of distributed objects, where the location of an object is determined by the hash-value of its name. For example, cryptographic hash-functions like MD5 [R92] or SHA1 [E01a] map arbitrary strings to 128-bit or 160-bit hash-values respectively. These can be used to map arbitrary object names into h-bit hash-values, where h depends upon the hash function being used. Without loss of generality, we assume that hashing maps object names onto the unit interval $\mathcal{I} = [0, 1)$. If a hash function maps an object to an h-bit hash-value H, then $H/2^h \in \mathcal{I}$.

In a DHT, each participating machine – a host on the Internet – is also assigned an ID in $\mathcal{I}$. At any instant, the current set of IDs defines the current set of partitions that the hash-table has been divided into. Each host is the manager of a distinct partition, being responsible for all objects whose names hash into that partition. Now, in order to

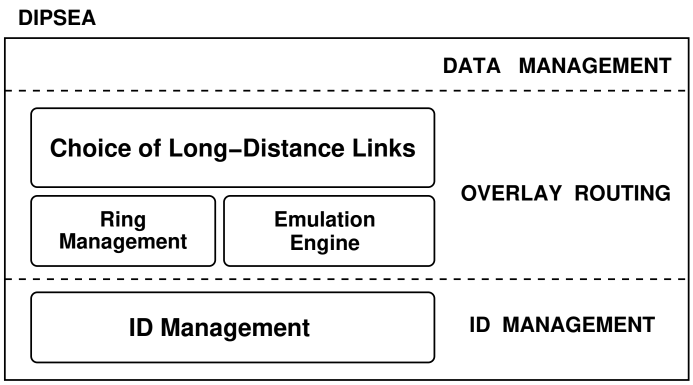


Figure 1.1: Dipsea: *A three-layered architecture for building Distributed Hash Tables. Efficient algorithms for ID Management are described in Chapters 2 and 3. See Section 1.3 for a brief description of Ring Management. See Chapter 4 for the Emulation Engine, and Chapters 5 through 9 for Choice of Long-Distance Links. Data Management is not discussed in this dissertation.*

insert, retrieve or update an object, we first compute the hash-value of its name, and then contact the manager whose partition includes that hash-value. In order to contact the manager, knowledge of its physical address – its IP address on the Internet – is required. This necessitates a mechanism for mapping the current set of partitions to the IP addresses of their corresponding managers. Decentralization dictates that such a mapping neither be maintained as global information nor be available at some central location. One possible solution is to make the managers establish *links* among themselves, as a function of their IDs. Taken together, these links constitute what is known as an *overlay routing network*. A request to insert, retrieve or update an object is handed over to the overlay which routes the request to the appropriate manager. Each manager along the route chooses one of its out-going links for forwarding the request.

1.1 The Design of Dipsea

Dipsea is a modular architecture for building Distributed Hash Tables (see Figure 1.1 for a block-diagram). The overall system is divided into three layers:

I. The ID Management Layer is responsible for assigning IDs to new managers (hosts on the Internet) that join the system. It also re-assigns the IDs of one or more existing hosts in response to arrivals and departures of hosts. Assignments and re-assignments of IDs should be automatic and decentralized. A good ID Management scheme is simple, low-cost in terms of network traffic, handles arrivals and departures of hosts, and ensures that the variation in partition sizes is minimal. The last requirement is motivated by load-balance, assuming a homogeneous set of hosts. We summarize our design of the ID Management layer in §1.2.

II. The Overlay Routing Layer is responsible for maintaining the links between hosts. The overlay network must satisfy several properties: small number of links per host, short routes between arbitrary hosts, low-latency routes, routing load-balance and resilience to host/link failures. The out-going links of a manager have to be re-configured in the face of arrivals and departures of other hosts, in a decentralized fashion.

    Each host in Dipsea has two kinds of links – *short-distance* and *long-distance* – maintained by three modules (all three are part of the Overlay Routing Layer):

    (a) The *Ring Management* module is responsible for maintaining the short-distance links, which are defined as follows. Imagine the managers along the circumference of a circle with unit perimeter, sorted by their IDs. Each manager then makes links with f successors (in the clockwise direction along the circle) and f predecessors (in the anti-clockwise direction along the circle), where f is a design parameter. Taken together, these short-distance links constitute a *fault-tolerant ring*. We discuss the Ring Management module further in §1.3.

    (b) The *Emulation Engine* absorbs the design complexity associated with maintenance of long-distance links caused due to (i) dynamism (arrival/departure of hosts), (ii) scale (variation in the average number of hosts over time) and (iii) concerns for physical network proximity. The Engine is responsible for handling the join/leave protocols *independent* of the specific family of routing networks being "mimicked" or "emulated" (*e.g.,* hypercubes or de Bruijn graphs). This point will become clearer in §1.4 where we explain the Emulation Engine in more detail.

    (c) *Choice of Long-Distance Links* is concerned with choosing the right family of networks for emulation. An important criterion for this choice is the trade-off between the number of links per manager and the average number of routing hops. We discuss our results pertaining to Choice of Long-Distance Links in §1.5 and §1.6.

The design philosophy separating the short- and the long-distance links is as follows: The short-distance links ensure *correctness* of the Overlay Routing layer. They guarantee that all managers are connected at all times and are able to communicate with each other. The long-distance links are for *efficiency* of the Overlay Routing layer. They ensure that routes between arbitrary pairs of managers require few hops, with each hop having low latency.

III. The Data Management Layer is responsible for durability and availability of objects which is achieved by a combination of replication and caching. Two schemes for replication have been proposed: (a) Replication of the contents of a manager at each of r successors along the circle, where r is a small integer, and (b) Erasure codes for breaking a large object into r' smaller-sized objects so that any $r < r'$ of the small-sized objects are sufficient to regenerate the original object, where r, r' are small integers. See Weatherspoon and Kubiatowicz [WK02] for a comparison between the two approaches. For caching protocols, see the design of Tapestry [ZHS+04], CUP [RB03] and Beehive [RS04a]. We do not discuss the Data Management Layer further in this dissertation.

The modularity of Dipsea imbues the overall system with several good properties:

1. A large problem has been broken into smaller sub-problems, each of which can be attacked more or less independently. This contributes to reduction in complexity — it is possible to explore the design space of a sub-problem in its entirety without being encumbered by its interactions with other sub-problems. For example, ID Management can be done independent of the family of routing networks we wish to emulate. Moreover, the complexity arising out of scale and dynamism is handled by the Emulation Engine, independent of the specific family that we emulate.

2. A modular design places existing DHT designs and improvements suggested for them into a common algorithmic framework. This helps us identify more clearly the commonalities and differences in various ID Management algorithms and routing networks.

3. The best solutions for individual sub-problems can be identified and put together to arrive at an overall design that is far more powerful than a design arrived at by a holistic approach. For example, by using the ID Management algorithms developed in Chapters 2 and 3 along with the Emulation Engine in Chapter 4, we are able to

plug-and-play arbitrary families of routing networks. Finally, by choosing any of the randomized routing networks studied in Chapters 7 or 8, we arrive at an overall design more efficient than any of the existing designs.

In the next five Sections, we describe the salient features of the following modules: ID Management in §1.2, Ring Management in §1.3, the Emulation Engine in §1.4, and Choice of Long-Distance Links in §1.5 and §1.6. For ID Management, the Emulation Engine and Choice of Long-Distance Links, we believe that our algorithms are the *best*-known.

1.2 ID Management

Consider a dynamic set of *managers* lying on the circumference of a circle. The managers divide the circumference into disjoint arcs. We will call each arc, a *partition*. Each manager manages those points that lie on the partition between itself and its clockwise successor along the circle. Three operations on the set of managers are allowed: (a) a new manager can be inserted into the set, (b) an existing manager can be deleted, and (c) an existing manager can be re-assigned to a new position on the circumference of the circle. In Chapters 2 and 3, we describe efficient *decentralized* algorithms for dividing the circle evenly among managers as they arrive and depart. We briefly sketch the key ideas here:

1. **Decentralization**: We model decentralization by making the assumption that a manager does not have global information – it does not know the IDs of all other managers at any time. However, it is possible to learn about the positions of other managers in two ways:

   (a) *Local Probe*: Using the short-distance links of the overlay routing network, it is possible for a manager to retrieve the IDs of k managers adjacent to a manager at the cost of $2k$ messages (assuming that the number of short-distance links per manager is $f = 1$). We call such a probe, a "Local Probe of size k".

   (b) *Random Probe*: Using the long-distance links of the overlay routing network, it is possible to route a message to the manager of a randomly-chosen point on the circle by paying a cost of R messages, with high probability[†]. The long-distance links in the earliest DHT overlay routing networks were based upon

[†]By "with high probability" (w.h.p.), we mean "with probability at least $1 - O(n^{-\lambda})$ for some constant $\lambda > 1$, for a system with n participants".

the hypercube and its variants. With n managers, these networks can route in $R = \Theta(\log n)$ messages, with only $\Theta(\log n)$ links per node. Later papers have shown that it is possible to achieve $R = \Theta(\log n / \log\log n)$ with the same number of long-distance links per manager. We discuss these results in detail in Chapter 8.

2. **Desiderata**: From a systems standpoint, a good ID management algorithm should possess each of the following properties: (a) the algorithm should be decentralized and simple, (b) the algorithm should entail low-cost, measured in terms of messages required for various probes, (c) the variation in partition sizes should be small to ensure load balance among managers, and (d) in response to arrivals and departures of managers, the number of ID re-assignments of existing managers should be minimal.

   We will quantify the variation in partition sizes by σ, defined as the ratio between the lengths of the largest and the smallest partitions.

3. **A Simple Scheme**: The following scheme was used by early DHT implementations. The scheme is decentralized and low-cost but results in highly uneven partition sizes.

   No Probes: Upon arrival, a manager places itself at randomly chosen point on the circle. No existing manager is re-assigned.

   The message complexity is zero, since no local or random probes are sent. However, the distribution of IDs results in $\sigma = \Theta(n \log n)$ (see King and Saia [KS04]).

4. **Our Contributions**: We have devised the following two schemes:

   (a) One Random Probe plus One Local Probe of size $(c \log n)$: An ID for a newly-arrived manager is derived as follows. We carry out one random probe to identify a manager, followed by a local probe of size $c \log n$ in the vicinity of that manager, and *split* the largest partition into two equal halves. In response to deletion of a randomly chosen manager, a local probe of size $c \log n$ is carried out in the vicinity of the departed manager and at most one existing manager is re-assigned. See Chapter 2 or reference [M04] for more details.

   The algorithm is the first one to enjoy all of the following properties: (a) both arrivals and departures of managers are handled, (b) departure of a manager causes at most one existing manager to change its ID, (c) the ratio of the largest to the smallest partition is at most 4, with high probability, and (d) the expected cost per arrival/departure is $\Theta(R + \log n)$ messages, where n denotes the current number of participants, and R denotes the cost of routing one message by using the long-distance links. Variations of the basic algorithm diminish the ratio between the largest and the smallest partition to $(1 + \epsilon)$, for any $\epsilon > 0$, albeit at the cost of $O(R + \frac{1}{\epsilon^2} \log n)$ messages and re-assignment of $O(\frac{1}{\epsilon})$ existing managers per arrival/departure. This is the first algorithm to allow such fine-tuning.

(b) r Random Probes plus r Local Probes of size v, with $(rv \geq c \log n)$: We carry out r random probes, followed by local probes of size v in the vicinity of the r managers. We then split the largest partition into two equal halves. The scheme is a generalization of the scheme above, but handles only arrivals of managers. An extension to the scheme that handles departures also is empirically shown to work well – we have not been able to formally analyze it yet.

The scheme guarantees $\sigma = \Theta(1)$ at the cost of $\Theta(rR + v)$ messages if each node maintains knowledge of v of its neighbors. When $R = o(\log n)$, the scheme is superior to previous schemes. In particular, when $R = \Theta(\log n / \log \log n)$, the optimal cost is $\Theta(\log n / \sqrt{\log \log n})$ messages per arrival or departure of hosts, by fixing $r = \Theta(\sqrt{\log \log n})$. See Chapter 3 or reference [KM04] for more details.

Notice that there are two criteria for measuring the efficacy of an ID Management algorithm: the total number of messages and the number of re-assignments of existing managers. For message complexity, we have ignored the messages required for establishing short-distance and long-distance links with other managers, once a manager has been assigned or re-assigned its ID. The philosophy underlying our accounting policy is as follows: The ID Management module is independent of the long-distance links – it treats long-distance links as a black-box that supports random probes. Therefore, messages associated with link-establishment are ascribed to the Overlay Routing Layer. The ID Management algorithm is not completely oblivious of these costs – it is quantified by the number of re-assignments required by the algorithm, which it strives to minimize.

5. **Choice of Long-Distance Links**: Both of the ID management algorithms described so far enjoy an important property: they assume the existence of short-distance links that support local probes, but are *independent of the long-distance links*, the union

of which is treated as a black-box supporting random probes. In contrast, a scheme by Adler *et al* [AHKV03] also guarantees $\sigma = \Theta(1)$ at the cost of $\Theta(\log n)$ messages. However, the scheme has been designed for a specific set of long-distance links: the hypercube. The idea is to identify a manager with one random probe, and to split the largest of this manager's long-distance neighbors, as defined by a hypercubic routing network. The advantage of having an ID management scheme that is independent of the long-distance links is that the two modules: ID Management and Choice of Long-distance Links (see Figure 1.1 on page 2) are cleanly separated.

1.3 Ring Management

Ring Management pertains to the maintenance of the short-distance links of the overlay routing network. Each host makes TCP connections with f successors and f predecessors along the circle, where f is a tunable parameter. The purpose of these links is to create a *fault-tolerant ring*. If $f = \Omega(\log n)$, even after the disappearance of half the nodes, chosen uniformly at random, the remainder of the nodes remain connected with high probability (see Liben-Nowell *et al* [LNBK02]). Practical schemes for Ring Management have been devised in the context of two system implementations (see Chord [SMK$^+$01] and Bamboo [RGRK04]). The primary challenge is to maintain the correct neighbor-sets or *views* of ring members in the face of frequent arrivals and departures of hosts. See Li *et al* [LMP04] for recent theoretical work on provably correct ring management protocols that handle concurrent joins and leaves. In this dissertation, we do not discuss Ring Management further.

1.4 The Emulation Engine

A common design goal is to make the long-distance links of the overlay routing network *mimic* or *look like* a well-known graph structure, *e.g.,* a hypercube, a butterfly network or a de Bruijn graph. These three basic graphs, along with several of their variants, have been well-studied by computer scientists since 1980's in the context of inter-connection networks for parallel machine architectures (see the classic book by Leighton [L92] for theoretical foundations of this area, and the book by Duato *et al* [DYN03] for practical design issues). Four challenges arise when we attempt to make the long-distance links mimic such graphs:

a) *Arbitrary Number of Nodes*: Hypercubes are defined for 2^k nodes, butterflies have $k2^k$ nodes, and de Bruijn graphs are defined for m^k nodes, where $k, m \geq 1$ are both integers. However, in a DHT, the current number of managers is not necessarily 2^k or $k2^k$ or m^k.

b) *Dynamism*: The set of nodes in a DHT changes over time as new managers arrive and existing managers depart. In contrast, the number of nodes in a parallel machine is fixed.

c) *Scale*: The number of nodes in a DHT exhibits large variation, spanning several orders of magnitude.

d) *Physical Network Proximity*: Since DHT nodes belong to different geographical regions of the world, the latency (or the ping-time) between a pair of randomly-chosen nodes is quite high. It is desirable that most long-distance links have low latency.

Recently, certain families of *random graphs* have also been investigated for routing purposes (see Chapters 8 and 9 for more details). Some of these are defined for arbitrary integers whereas others are defined over successive powers of two. The same four challenges, as listed above, arise if we desire that the long-distance links of the overlay routing network mimic one of these random graphs.

On the whole, the problem of mimicking a given family of graphs, deterministic or randomized, can succinctly be stated from the perspective of a manager:

> DEFINITION (The Problem of Scalable and Dynamic Emulation of Network Families). *Assume that we would like to mimic a specific family of graphs, say the hypercube. Given a manager with a specific ID, which other managers should it make long-distance links with?*

The Emulation Engine solves the above problem. It is described in detail in Chapter 4. Here, we briefly sketch the key ideas:

1. **ID Distributions**: The Emulation Engine handles two different distributions of IDs:

    (a) RANDOM distribution of IDs: Each manager chooses an ID independently and uniformly at random from $\mathcal{I} = [0, 1)$.

    (b) BALANCED distribution of IDs: Manager IDs correspond to leaf nodes of a binary tree whose leaf nodes belong to at most three different levels: $[\log_2 n]$ and $[\log_2 n] \pm 1$, where $[x]$ denotes the integer closest to x.

RANDOM distribution of IDs results from the No Probes scheme (see Section 1.2) – the resulting distribution of partition sizes is highly skewed. BALANCED distribution of IDs results from the following two algorithms: "One Random Probe plus One Local Probe of size $(c \log n)$" and "r Random Probes plus r Local Probes of size v, with $(rv \geq c \log n)$". In both of these algorithms, IDs correspond to leaf nodes of a binary tree in which each internal node has exactly two children. If we label the left and right branches of internal nodes with 0 and 1 respectively, then the sequence of bits from the root to a leaf node, treated as a fraction in $\mathcal{I}$, constitutes the ID associated with that leaf. The resulting distribution of IDs is BALANCED.

2. **Link Establishment**: Let $\langle G_0, G_1, G_2, \ldots \rangle$ denote an infinite family of directed graphs where graph G_i is defined over 2^i nodes. Let $C(\mathbf{x})$ denote a *cluster* consisting of all managers whose IDs have prefix $\mathbf{x}$. Consider a manager with an ℓ-bit ID $\mathbf{x}$. It establishes three sets of links: one set corresponding to $\mathbf{x}_1$, the $(\ell - 2)$-bit prefix of its ID, another set corresponding to $\mathbf{x}_2$, the $(\ell - 3)$-bit prefix of its ID, and finally, a set corresponding to $\mathbf{x}_3$, the $(\ell - 4)$-bit prefix of its ID. Let $\mathbf{x}_1^1, \mathbf{x}_1^2, \ldots, \mathbf{x}_1^{i_1}$ denote the i_1 neighbors of label $\mathbf{x}_1$ in graph $G_{\ell-2}$. Let $\mathbf{x}_2^1, \mathbf{x}_2^2, \ldots, \mathbf{x}_2^{i_2}$ denote the i_2 neighbors of label $\mathbf{x}_2$ in graph $G_{\ell-3}$. Let $\mathbf{x}_3^1, \mathbf{x}_3^2, \ldots, \mathbf{x}_3^{i_3}$ denote the i_3 neighbors of label $\mathbf{x}_3$ in graph $G_{\ell-4}$. Then node $\mathbf{x}$ makes $i_1 + i_2 + i_3$ links with one member each of the following clusters: $C(\mathbf{x}_1^1), C(\mathbf{x}_1^2), \ldots, C(\mathbf{x}_1^{i_1})$, $C(\mathbf{x}_2^1), C(\mathbf{x}_2^2), \ldots, C(\mathbf{x}_2^{i_2})$, and $C(\mathbf{x}_3^1), C(\mathbf{x}_3^2), \ldots, C(\mathbf{x}_3^{i_3})$. For the other end of a link, any member of the destination cluster suffices. For example, a manager with ID 0.010111 would make links corresponding to $B_1 = 0.0101$, $B_2 = 0.010$ and $B_3 = 0.01$. When emulating hypercubes, links would be established with one member each of clusters whose prefixes are listed below:

$$\begin{array}{c} 0.1101 \quad 0.0001 \quad 0.0111 \quad 0.0100 \\ 0.110 \quad 0.000 \quad 0.011 \\ 0.11 \quad 0.00 \end{array}$$

3. **Routing Protocol**: Let us label each node with a triplet of integers, corresponding to the three sets of links it makes. Thus a node with an ℓ-bit ID is labeled with $\langle \ell - 2, \ell - 3, \ell - 4 \rangle$. Routing starts off along that set of links that correspond to the smallest integer in the triplet of the source node. Routing switches to links corresponding to next higher integer if it encounters a node which is labeled with a different triplet. In

BALANCED distribution of IDs, IDs correspond to leaf nodes in a binary tree at levels $[\log_2 n]$ or $[\log_2 n] \pm 1$, where n is the total number of leaf nodes, and $[x]$ denotes the integer closest to real number x. Therefore, each cluster on $[\log_2 n] - 1$ or fewer bits is non-empty. Also, there are at most three different triplets used in labeling all the nodes, and it is guaranteed that the integer $[\log_2 n] - 1$ is a member of all the triplets. Therefore, a message will eventually be delivered to a cluster on $[\log_2 n] - 1$ bits. At this point, the remaining distance is $\Theta(1)$ hops along the circle, which can be covered by using the short-distance links.

4. **Network Proximity Awareness**: It is important that each of the long-distance links have low latency in terms of inter-host IP ping times. Instead of making a link with an arbitrary host belonging to the destination cluster, if we were to make a link with the *closest* host, as per the ping-times, we would expect small ping times on average. In Chapter 4, we show that indeed, for BALANCED distribution of IDs, by making links corresponding to $\ell - 3$, $\ell - 4$ and $\ell - 5$ bits, we can ensure that most clusters have sixteen or more hosts, which is sufficient to guarantee small inter-host IP ping times. We validate our design through a series of experiments using real-world latencies measured by the Skitter project [Ski] and by using the GT-ITM topology generator [ZCB96].

Notes:

1. Our approach for incorporating network proximity awareness into long-distance links is unique because of its generality. Previous work has focused on specific topologies like Chord [GGG+03, ZGG03, DLS+04] or hypercubes [GGG+03, CDHR03, RGRK04]. In fact, we show that network proximity can be factored into the design *independent* of the choice of long-distance links, in a generic fashion.
2. *Non-Power of Two Families*: We transform the given family of graphs into another family defined over powers of two. Then the emulation technique developed so far is readily applicable.
3. It is possible to carry out emulation with only *two* set of links per node instead of three, for both RANDOM and BALANCED distributions of IDs. See Chapter 4 for more details.
4. The Emulation Engine absorbs complexity arising out of dynamism (arrivals/departure of hosts), scale (variation in the average number of hosts), and concerns of physical network proximity. This leaves us free to explore *static* networks defined over successive

powers of two. These families of networks constitute the top-most module of Dipsea: Choice of Long-Distance Links (see Figure 1.1 on page 2). We devote Chapters 5 through 9 in understanding that module.

1.5 Choice of Long-Distance Links: Deterministic

What is a good family of networks for making the long-distance links? An important trade-off is between the number of links per node and length of routes. The Degree-Diameter Problem, studied in extremal graph theory, seeks to identify the graph with the maximum nodes whose diameter is Δ, with each node having out-degree at most d (see Delorme [D04] for a survey). A well-known upper bound for the number of nodes is $1+d+d^2+\cdots+d^{\Delta} = \frac{d^{\Delta+1}-1}{d-1}$, also known as the Moore bound. A general lower bound is $d^{\Delta} + d^{\Delta-1}$, achieved by Kautz digraphs [K68, K69], which are slightly superior to de Bruijn graphs [dB46] whose size is only d^{Δ}. Two consequences of these results are: (a) with out-degree d per node, the diameter of any graph on n nodes is $\Omega(\log n/\log d)$, and (b) there exist constructions (high-degree butterfly networks and de Bruijn graphs, for example) whose diameter is $O(\log n/\log d)$. Such graphs have been studied extensively in the context of routing in parallel machine architectures (see the book by Leighton [L92]).

We make two contributions in the space of deterministic routing networks:

1. **Optimal Routing in Chord**

   In Chapter 5, we characterize shortest paths in the following graph:

   DEFINITION (Chord). *Consider an undirected graph on 2^b nodes arranged in a circle. Nodes are labeled with b-bit identifiers from 0 through 2^b-1 going clockwise. An edge (x,y) exists iff x and y are 2^k positions apart on the circle for some $k \geq 0$, i.e., $|x-y|$ equals either 2^k or 2^b-2^k for some $0 \leq k < b$.*

   From the perspective of shortest paths, the definition of Chord is deceptively simple; it hides a rich combinatorial structure, some of which we unearth in Chapter 5.

   In the standard Chord routing algorithm [SMK+01], messages are forwarded along only those edges that diminish the clockwise distance by some power of two. Routing is clockwise and greedy, never overshooting the destination. If a message is destined for a node that is clockwise distance d away, routing is equivalent to performing left-to-right bit-fixing to convert the 1s in the binary representation of d to zero. For example, if d is 14 (1110 in binary), the standard Chord routing algorithm uses steps of 8, 4 and 2 in that order, thus converting the leftmost 1 in the remaining distance to a 0 at each step. The longest path has length b and the average path length is $b/2$.

We show that shortest paths in Chord have a strong connection with the *Binary Subtraction Problem*: Given a positive integer d, find a pair of non-negative integers $\langle d', d'' \rangle$ such that the number of 1-bits in d' and d'' is minimal, subject to the constraint $d = d' - d''$. For example, the shortest route to cover clockwise distance 14 (1110 in binary) is to use a clockwise step of 16 in combination with an anti-clockwise step of length 2, which can be seen as an optimal way of expressing 14 as the difference of two numbers. We solve the Binary Subtraction Problem, presenting a non-deterministic procedure that generates all the optimal solutions. This enables us to identify optimal routes between any pair of nodes in Chord. We show that Chord's diameter is $\lfloor b/2 \rfloor$. However, the average all-pairs shortest-path length is only $b/3 + \Theta(1)$. Interestingly, two simple algorithms for computing optimal routes can be encoded compactly by finite-state automata. The average shortest-path lengths are then computed by treating the automata as Markov Chains. Finally, we extend our results to higher-base versions of Chord.

2. **Greedy Routing on a Circle**

Consider n nodes placed in a circle, labeled 0 through $n - 1$. GREEDY routing, as formally defined below, is a natural routing strategy: a node forwards a message along that out-going link that minimizes the *distance* remaining to the destination:

DEFINITION (Greedy Routing). *In graph (V, E) with distance function $\delta : V \times V \rightarrow \mathcal{R}^+$,* GREEDY *routing entails the following decision: Given a target node t, a node u with neighbors $N(u)$ forwards a message to its neighbor $v \in N(u)$ such that $\delta(v, t) = \min_{x \in N(u)} \delta(x, t)$.*

For graphs consisting of n nodes placed in a circle, two natural distance metrics are the clockwise-distance and the absolute-distance between pairs of nodes.

$$\delta_{clockwise}(u,v) = \begin{cases} v-u & v \geq u \\ n+v-u & \text{otherwise} \end{cases}$$

$$\delta_{absolute}(u,v) = \begin{cases} \min\{v-u, n+u-v\} & v \geq u \\ \min\{u-v, n+v-u\} & \text{otherwise} \end{cases}$$

In Chapter 6, we study the following combinatorial problem:

I Given integers d and Δ, what is the largest graph that satisfies two constraints: the out-degree of any node is at most d, and the length of the longest GREEDY route is at most Δ hops?

II Given integers d and n, design a network in which each node has out-degree at most d such that the length of the longest GREEDY route is minimized.

We construct Papillon, a family of graphs that offers optimal trade-off between out-degree per node and worst-case route lengths using GREEDY routing. We have two constructions for Papillon, one for distance function $\delta_{clockwise}$ and another for distance function $\delta_{absolute}$. Both families are variants of butterfly networks:

(a) A network with $n = \kappa^m m$ nodes, each with κ links per node, and GREEDY routes of length at most $3m-2$ (at most $2m-1$ on average), with $\delta_{clockwise}$ as the distance-function.

(b) A network with $n = (2k+1)^m m$ nodes, $2k+2$ links per node, and GREEDY routes of length at most $3m-2$ (at most $2m-1$ on average) with $\delta_{absolute}$ as the distance-function.

In terms of d and Δ, Papillon has $n = d^{O(\Delta)} \cdot \Delta$ nodes. As long as $m = O(poly(k))$, route lengths are $\Theta(\log n/\log k)$, which is asymptotically optimal, given n and k. In particular, if $k = O(\log n)$, route lengths are $\Theta(\log n/\log\log n)$. Papillon is the first construction that achieves such optimality for distance functions $\delta_{clockwise}$ and $\delta_{absolute}$. Curiously, in both networks, GREEDY routing does not route along shortest paths. We show this constructively by identifying routes which are shorter than those afforded by GREEDY routing. These routes also guarantee uniform edge congestion.

1.6 Choice of Long-Distance Links: Randomized

Several new families of *randomized* routing networks have been proposed in the context of DHTs. All of these networks are defined for n nodes placed in a circle, with nodes labeled 0 through $n-1$ in the clockwise direction. Each node is connected with its successor and predecessor along the circle. Each node also makes one or more *long-distance* links with other nodes.

An important distinction between deterministic and randomized routing networks pertains to knowledge of the overall graph structure. In a deterministic routing network, we assume that the structure of the network is global information. Therefore, messages can be sent along shortest paths. In a randomized routing network, each node makes links as a function of some random bits generated locally. We assume that these random bits are not global information. Therefore, it is not possible to send messages along shortest paths, in general. This motivates the need for *decentralized routing strategies* which allow a node to forward messages on the basis of as little knowledge of other nodes' random bits as possible. GREEDY routing, as defined earlier, is a natural decentralized routing strategy.

We make three contributions in the space of randomized routing networks:

1. **Symphony: Routing in a Small-World**

   In Chapter 7, we develop Symphony, a randomized routing network. Symphony is an adaptation of Kleinberg's small-world construction [K00] in one dimension. Consider n nodes lying on the circumference of a circle, labeled 0 through $n-1$. Each node establishes a *short-distance* link with its immediate neighbor along the circle. Node $\mathbf{x}$ establishes $k \geq 1$ *long-distance* links as follows: For each link, node $\mathbf{x}$ first draws a random number r from the probability distribution $p(x) = 1/(x \ln n)$ where $x \in [1, n]$ and then establishes a link with node $\lceil \mathbf{x} + r \rceil \bmod n$.

   With $k \leq \log n$ links per node, GREEDY routing with distance function $\delta_{clockwise}$ in Symphony requires $O(\frac{1}{k}\log^2 n)$ hops on average. We also study a variant of GREEDY routing:

DEFINITION (Greedy with 1-Lookahead Routing). *In graph (V, E) with distance function $\delta : V \times V \to \mathcal{R}^+$,* GREEDY *with 1-*LOOKAHEAD *routing entails the following decision: A node takes its neighbor's neighbors also into account when making routing decisions. Let $N(x)$ denote the neighbors of node x. Given target node t, node u first identifies node z such that $\delta(z,t) = \min_{x \in N(u)}\{\delta(x,t), \min_{y \in N(x)} \delta(y,t)\}$. If link (u, z) exists, then node u forwards the message to node z. Otherwise, node v exists such that both (u, v) and (v, z) exist; u forwards the message to v, which then forwards the message to z.*

Experiments indicate that GREEDY with 1-LOOKAHEAD reduces average route length by about 40% when $n = 2^{15}$ nodes. This observation motivated further theoretical analysis of GREEDY with/without 1-LOOKAHEAD in Symphony, leading to results developed in Chapter 8.

2. **Greedy with/without Lookahead in Randomized Routing Networks**

   In Chapter 8, we study a variety of randomized routing networks. All networks defined below are directed graphs with $n = 2^\ell$ nodes labeled 0 through $n-1$, arranged in a circle. Each node is connected to its successor by a *short-distance* link. The rest of the links are said to be *long-distance* and involve random choices.

   ☆ Randomized-Hypercube [CDHR03, GGG+03]

   The out-degree of each node is ℓ. For each $1 \le i \le \ell$, node $\mathbf{x}$ makes a link with node $\mathbf{y}$ defined as follows: The top $i-1$ bits of $\mathbf{y}$ are identical to those of $\mathbf{x}$. The i^{th} bit is flipped. Each of the remaining $\ell - i$ bits is chosen uniformly at random. The distance-function for routing is δ_{xor}, which is defined as $\delta_{xor}(u, v) = |u \oplus v|$ for nodes u and v, the Hamming distance between the labels of the two nodes.

   ☆ Randomized-Chord [ZGG03, GGG+03]

   Node $\mathbf{x}$ makes ℓ links as follows: Let $r(i)$ denote an integer chosen uniformly at random from the interval $[0, 2^i)$. Then for each $0 \le i < \ell$, node $\mathbf{x}$ creates an edge with node $(\mathbf{x} + 2^i + r(i)) \bmod n$. Each node has out-degree ℓ. The distance-function for routing is $\delta_{clockwise}$.

   ☆ Symphony [MBR03]

   Node $\mathbf{x}$ establishes $k \ge 1$ *long-distance* links as follows: For each link, node $\mathbf{x}$ first draws a random number r from the probability distribution $p(x) = 1/(x \ln n)$

where $x \in [1, n]$ and then establishes a link with node $\lceil \mathbf{x} + r \rceil \bmod n$. The distance-function for routing is $\delta_{clockwise}$.

DEFINITION (Average Route Length $R(n)$). *Let $r(\mathbf{x}, \mathbf{y})$ denotes the length of the route from node $\mathbf{x}$ to node $\mathbf{y}$. For deterministic graphs like Chord and hypercube, $R(n) \equiv n^{-2} \sum_{\mathbf{x},\mathbf{y} \in [0,n-1]} r(\mathbf{x}, \mathbf{y})$. For randomized graphs, $R(n) \equiv n^{-2} \mathbf{E} \sum_{\mathbf{x},\mathbf{y} \in [0,n-1]} r(\mathbf{x}, \mathbf{y})$.*

The following picture emerges in Chapter 8:

A) *Deterministic topologies – the hypercube and Chord – have diameter* $\Theta(\log n)$.

In fact, GREEDY routing with distance function δ_{xor} is optimal for the hypercube, and GREEDY routing with distance function $\delta_{absolute}$ is optimal for Chord. Both route along shortest paths, with $R(n) = \Theta(\log n)$. 1-LOOKAHEAD offers no improvement.

B) *Randomization reduces the diameter to* $\Theta(\log n / \log \log n)$ *in expectation.*

Each of the following networks has diameter $\Theta(\log n / \log \log n)$: Randomized-Chord, Randomized-Hypercube, and Symphony with $k = \Theta(\log n)$ links per node. In contrast, the deterministic topologies (Hypercube and Chord) have diameter $\Theta(\log n)$.

A small diameter does not necessarily mean that there exist efficient decentralized routing algorithms. This motivates a formal analysis of GREEDY with/without 1-LOOKAHEAD:

C) GREEDY *routing is unable to discover optimal routes in randomized networks.*

GREEDY routing requires $R(n) = \Theta(\log n)$ hops on average for each of the following randomized networks: Randomized-Chord, Randomized-Hypercube, and Symphony with $\Theta(\log n)$ links per node.

D) GREEDY *with 1-*LOOKAHEAD *is asymptotically optimal for randomized networks.*

Each of the following randomized networks requires $R(n) = \Theta(\log n / \log \log n)$ hops on average with GREEDY with 1-LOOKAHEAD routing: Randomized-Chord, Randomized-Hypercube, and Symphony with $\Theta(\log n)$ links per node.

E) *Simulations show that the average route length of* GREEDY *with 1-*LOOKAHEAD *in Randomized-Chord, Randomized-Hypercube and Symphony with* $\log_2 n$ *links per* *node is within 10% of average route lengths in de Bruijn graphs with as many links per node.*

Results B), C) and D) hold for three more randomized networks: SkipNet [HJS+03], skip-graphs [AS03] and small-world Percolation Networks (see reference [MNW04]). Additional results proved in Chapter 8 include the following:

a) With k links per node, $R(n) = \Omega(\frac{1}{k}\log^2 n)$ hops for GREEDY routing in Symphony.

b) With k links per node, $R(n) = O(\log^2 n/(k \log k))$ hops for GREEDY with 1-LOOKAHEAD routing in Symphony.

We also study the following randomized networks:

- ⋆ Sparse-Chord [M03]

  In Chord, a node makes $\ell - 1$ long-distance links with other nodes at the following clockwise-distances: $\langle \frac{n}{2}, \frac{n}{4}, \frac{n}{8}, \ldots, 4, 2\rangle$. In Sparse-Chord, each node chooses $k \geq 1$ out of these out-going links at random.

- ⋆ Sparse-Hypercube [M03]

  In a hypercube, a node makes ℓ long-distance links with other nodes corresponding to bit-flips in each of ℓ positions of its label written in binary. In Sparse-Hypercube, each node chooses $k \geq 1$ out of these out-going links at random.

Using the Bit-Collection protocol (see reference [M03] or Chapter 8 for more details), $R(n) = O(\log n)$ hops for both Sparse-Chord and Sparse-Hypercube, when $k = \Theta(\log \log n)$.

3. **Mariposa: A Randomized Butterfly**

   Mariposa is a randomized routing network which differs from those studied so far in terms of its design philosophy. The difference lies in whether a node learns about the random choices made by other nodes *before* or *after* link-establishment. We explain the point in more detail below.

   In Randomized-Chord, Randomized-Hypercube, Symphony, skip-graphs and SkipNet, each node generates some random bits locally, and establishes links with other nodes on the basis of the bits it generates. *After* link-establishment, a node can inspect the random bits of other nodes (typically, its neighbors with whom it has established links) for making good routing decisions. For example, GREEDY with 1-LOOKAHEAD follows this paradigm.

In Mariposa, a node generates some random bits. It then inspects the random bits of a few other nodes *before* it establishes its long-distance links. The knowledge gained by inspecting other nodes' random bits is used for making good decision for link-establishment itself.

In a nutshell, the routing networks we have studied so far use following sequence of operations:

- ➢ Generation of local random bits.
- ➢ Establishment of long-distance links.
- ➢ Inspection of non-local random bits for routing.

Mariposa uses the following sequence of operations:

- ➢ Generation of local random bits.
- ➢ Inspection of non-local random bits for establishment of long-distance links.
- ➢ Routing.

Mariposa is an interesting combination of butterfly networks and Kleinberg's small-world construction [K00]. With $3\ell + 3$ out-going links per node, Mariposa routes in $O(\log n / \log \ell)$ hops in the worst-case, which is asymptotically optimal. The construction improves upon Viceroy [MNR02], which also follows the sequence of operations outlined above. Viceroy routes in $O(\log n)$ hops in expectation with $\Theta(1)$ out-going links per node. Mariposa improves upon Viceroy in terms of the trade-off between out-degree and the *worst-case* length of routes when the out-degree is $\omega(1)$.

1.7 Peer-to-Peer Computing: A Historical Perspective

Distributed Computing has witnessed many a shift in direction, driven by increases in number, performance and connectivity of computers. Research into the subject was spawned by the 4-node ARPANET in 1969. Subsequent developments in inter-networking hardware, most notably the Ethernet in 1973, gave distributed systems research a big impetus. LAN-based systems were investigated in the 1980s leading to the development of *client-server* *systems* typified by NFS and HTTP servers. These systems enjoyed great success, leading to rapid deployment of the World Wide Web in the 1990s. As a consequence, research in the 1990s was dominated by web-based front ends and cluster-based back ends. The 1990s witnessed (a) the proliferation of computers that communicate over wide-area networks, and (b) reduction in the gap between performance+capacity of client- and server-class machines. As a result, the early 2000s witnessed large-scale distributed applications that treated all participants as *peers*. Examples of such systems are SETI@home [ACLW02], Freenet [CHM+02], Gnutella [RFI02, SGG02] and Kazaa [GDS+03]. The advent of such large-scale decentralized systems, also known as peer-to-peer systems, marks a departure away from the traditional client-server paradigm.

Since 2001, peer-to-peer systems have witnessed an explosive growth of academic interest. Broadly speaking, two categories of systems are being investigated: *structured* and *unstructured* – both systems have their pros and cons. Unstructured systems have already enjoyed great success in the form of file-sharing applications like Napster, Gnutella and Kazaa which are used by millions of users. These networks are unstructured in the sense that the set of links between machines are not dictated by some pre-defined topology. The system offers no performance guarantees; it works on a best-effort basis. However, the system supports complex queries and remains functional despite frequent arrivals and departures of heterogeneous participants. For a discussion of unstructured P2P systems, see Yang and Garcia-Molina [YGM01] and Chawathe *et al* [CRB+03]. For music-sharing applications, where imprecise/partial results to queries are acceptable, unstructured systems are sufficient. For applications that mandate stronger guarantees on data storage and retrieval, the systems community is investigating the design-space of *structured* P2P systems. These systems are harder to engineer because of the stronger guarantees on performance and correctness that the system is expected to deliver. So far, these systems have not witnessed large-scale deployments although the academia has investigated several potential applications. These include persistent data storage (OceanStore [KBC+00], Cooperative File System [DKK+01], Farsite [BDET00], and PAST [RD01b]), DNS (CoDoNS [RS04b]), resource discovery (SETS [BMR03]), cooperative web caching (Squirrel [IRD02]), and event notification with application level multicast (Bayeux [ZZJ+01], Scribe [RKCD01] and CAN-based Multicast [RHKS01]). Several of these applications have no centralized components and use a scalable DHT as a substrate.

Distributed Hash Tables: A Brief History

Distributed Hash Tables over clusters of machines have been extensively studied by the SDDS (Scalable Distributed Data Structures) community in the 90's. The term was coined in a seminal paper by Litwin, Niemat and Shneider [LNS96]. Gribble *et al* [GBHC00] implemented a highly scalable, fault tolerant and available SDDS on a cluster.

Distributed Hash Tables over thousands of machines that span wide-area networks were first investigated in early 2000s, when the first proposals appeared: CAN [RFHK01], Chord [SMK+01], Pastry [RD01a], P-Grid [A01] and Tapestry [ZHS+04]. The routing network of CAN is an adaptation of multi-dimensional tori. The routing scheme in Pastry, P-Grid and Tapestry shares similarities with an earlier prefix-based routing scheme due to Plaxton *et al* [PRR99]. Chord is a variation on hypercubes. All of these allow a manager (a machine/host on the Internet) to choose a random number in $\mathcal{I}$ as its ID. As a function of its ID, a manager makes links with other managers such that the union of the links approximates a hypercube. A hypercube is attractive because it is conceptually simple, and it has been well-studied (see Leighton [L92]). In an n-node network, the length of routes in a hypercube is $O(\log n)$ hops at the cost of only $O(\log n)$ out-going links per machine. Recently, some new implementations of DHTs have surfaced: Chord [DLS+04], Bamboo [RGRK04] and P-Grid [ACMD+03].

Since 2001, several improvements to the DHT design have been proposed:

◇ **Routing Networks**

A variety of graphs have been proposed for building the overlay routing network. These include high-degree de Bruijn graphs, as noted by several groups [AAA+03, FG03, KK03, LKRG03, NW03], multi-dimensional grids [RFHK01], and high-degree butterflies [KMXY03]. Most of these graphs have been well-studied in the context of routing in parallel machines (see the classic book by Leighton [L92] for theoretical foundations of this area). Interestingly, a variety of novel *randomized* routing networks have also been designed for DHT routing. These include Viceroy [MNR02] (a randomized butterfly network), Symphony [MBR03] (an adaptation of Kleinberg's small-world construction [K00]), Mariposa [M03] (another randomized butterfly network), randomized-Chord [ZGG03, GGG+03], randomized-hypercubes [CDHR03, GGG+03], skip-graphs [AS03] and SkipNet [HJS+03, HM03a]. The last two networks are adaptations of skip-lists (see Pugh [P90]).

◇ **ID Management and Load Balance**

Early DHT implementations allowed a participating machine to use a random number in $\mathcal{I}$ as its ID. A problem with this scheme is that some partition sizes are too small whereas others are too big. With n machines, the ratio between the sizes of the largest and the smallest partitions is $\Theta(n \log n)$ (see King and Saia [KS04]). In a homogeneous system, it is desirable that the variation in partition sizes be minimal. With this goal in mind, efficient decentralized ID management algorithms have been developed so that the partition balance ratio is as small as $\Theta(1)$. For example, Adler *et al* [AHKV03] have devised a scheme tailored for managers connected as a hypercube. Karger and Ruhl [KR04] have devised a scheme for Chord. Naor and Wieder [NW03] and Abraham *et al* [AAA$^+$03] have developed schemes which are independent of the routing network.

◇ **Physical Network Proximity**

Since participating machines belong to different geographical regions of the world, the latency (or the ping-time) between a pair of randomly-chosen participants is quite high. If most of the links in the overlay routing network are high-latency, and if routes are $O(\log n)$ hops long (as in a hypercube, for example), then the total time taken to transmit any message to its destination would be quite large. Such high latencies would render the DHT practically unusable. The problem of physical network proximity has been ameliorated for two specific routing networks: Chord and the hypercube. In both cases, the basic topology is altered by introducing randomization: Chord gets transformed into randomized-Chord [GGG$^+$03, ZGG03], and the hypercube gets transformed into randomized-hypercube [GGG$^+$03, CDHR03]. The key idea is to make the topology less rigid by introducing choices for every link that is established. This allows a manager to establish a link with the closest manager, from among the choices available.

Several questions emerge:

1. **ID Management**: What are the commonalities and differences in the various ID management schemes? Can the scheme developed by Karger and Ruhl [KR04] be used for routing networks other than Chord? Does each routing network engender its own ID management scheme (for example, the scheme by Adler *et al* [AHKV03] is

tailored for a hypercubic routing network)? Or is there a generic scheme that can be employed for arbitrary routing networks?

2. **Routing Networks**: What are the relationships between various deterministic and randomized routing networks? Are randomized routing networks better or worse than deterministic routing networks? Do join/leave protocols for different routing networks have anything in common? Finally, how do different routing networks compare with each other, when it comes to DHT routing?

3. **Physical Network Proximity**: Can physical network proximity be incorporated into arbitrary graph topologies, like butterflies and de Bruijn graphs? How do we introduce choices for making links in randomized routing networks? Do we have to handle each routing network on a case-by-case basis, or is there a generic scheme that suffices for all networks?

In a nutshell, the problem is that of identifying the right abstractions for building DHTs. The building-blocks for different abstractions should fit together snugly, leading to a modular system-design. Dipsea is a design in response to this need – it is a modular architecture for building DHTs. The overall design has been broken into modules which are more or less independent of each other. For each module, Dipsea has arguably the best-known design. By putting together the different modules, we arrive at an overall design of Dipsea that is more efficient than any of the existing implementations.

Remark: A DHT can also be used for storing just pointers to objects instead of the objects themselves. When used in this fashion, a DHT functions as a "Distributed Object Location Service" – a directory service much like Grapevine [BLNS82], DNS [MD88], or the Corba Name Server [V97]. A DHT-based system for DNS is being designed as CoDoNS [RS04b].

1.8 Dissertation Road-map

Each chapter in this dissertation is self-contained. ID Management is discussed in Chapters 2 and 3. The Emulation Engine is covered in Chapter 4. Two problems pertaining to deterministic randomized networks are solved in Chapters 5 and 6. Randomized routing networks are analyzed in Chapters 7 and 9. We present a summary along with directions for further research in Chapter 10.

⋆ **Chapter 2 (Balanced Binary Trees)**

We explore the "One Random Probe plus One Local Probe of size $(c \log n)$" scheme and its variations for ID Management. The analysis has also been published as

[M04] G S MANKU, Balanced Binary Trees for ID Management and Load Balance in Distributed Hash Tables, *Proc. 23rd ACM Symposium on Principles of Distributed Computing (PODC 2004)*, July 2004.

⋆ **Chapter 3 (Coupon Collection over Cliques)**

We discuss the "r Random Probes plus r Local Probes of size v, with $(rv \geq c \log n)$" scheme for ID Management. This is a generalization of the above scheme and requires a novel proof technique. We first analyze the following random process:

> Consider n/b bins, each of capacity b. All bins are initially empty. At successive trials, we choose a bins uniformly at random. If at least one of the chosen bins is non-full, we pick one of the non-full bins (from among the ones chosen) and place a ball into it.

We show that if $ab \geq c \log n$, then each of the first $\Omega(n)$ trials succeeds in placing a ball into some bin, and that all bins are full in $O(n)$ trials. These results, along with ideas borrowed from reference [AHKV03], are used to analyze the ID management scheme. These results have also been published as:

[KM04] K KENTHAPADI and G S MANKU, Structured Coupon Collection over Cliques for P2P Load Balance, *Manuscript*, Available as DB Group TR 2004-38, Computer Science Department, Stanford University, June 2004.

⋆ **Chapter 4 (Scalable and Dynamic Emulation of Network Families)**

The Emulation Engine is described in this Chapter. The engine enables "plug and play" of various families of routing networks, not only deterministic parallel interconnection networks but also the recently-discovered families of randomized routing networks. Some of these results have also appeared in the following publications:

[M03] G S MANKU, Routing Networks for Distributed Hash Tables, *Proc. 22nd ACM Symposium on Principles of Distributed Computing (PODC 2003)*, p 133–142, July 2003.

[M04] G S MANKU, Balanced Binary Trees for ID Management and Load Balance in Distributed Hash Tables, *Proc. 23rd ACM Symposium on Principles of Distributed Computing (PODC 2004)*, July 2004.

⋆ **Chapter 5 (Shortest Paths in Chord)**

The results have also appeared in the following paper:

[GM04] P GANESAN and G S MANKU, Optimal Routing in Chord, *Proc. 15th ACM-SIAM Symposium on Discrete Algorithms (SODA 2004)*, p 133–142, Jan 2004.

⋆ **Chapter 6 (Papillon: Greedy Routing on a Circle)**

We describe two variants of butterfly networks such that GREEDY routing with distance functions $\delta_{clockwise}$ and $\delta_{absolute}$ requires $O(\log n / \log d)$ hops in the worst-case, with d out-going links per node in an n-node network. The results have appeared in the following paper:

[AMM04] I ABRAHAM and D MALKHI and G S MANKU, The Degree-Diameter Greedy Routing Problem, *Manuscript*, July 2004.

⋆ **Chapter 7 (Symphony: Routing in a Small World)**

We develop Symphony, one of the first randomized routing networks proposed in literature. Symphony is an adaptation of Kleinberg's small-world construction [K00] in one dimension. We show that with $k \leq \log n$ links per node, GREEDY routing with distance function $\delta_{clockwise}$ takes $O(\frac{1}{k}\log^2 n)$ hops on average. Experiments indicate that GREEDY with 1-LOOKAHEAD reduces average route length by about 40% when $n = 2^{15}$ nodes. This observation motivated a theoretical analysis of GREEDY with/without 1-LOOKAHEAD in Symphony, leading to results developed in Chapter 8. Symphony is described in the following paper:

[MBR03] G S MANKU, M BAWA and P RAGHAVAN, Symphony: Distributed Hashing in a Small World, *Proc. 4th USENIX Symposium on Internet Technologies and Systems (USITS 2003)*, p 127–140, 2003.

⋆ **Chapter 8 (Greedy Routing with Lookahead)**

We study GREEDY routing with/without 1-LOOKAHEAD in Symphony, Randomized-Chord and Randomized-Hypercube. Although the definitions of these networks may appear different, the networks share significant structural similarities. The analysis has appeared previously as:

[MNW04] G S MANKU, M NAOR and U WIEDER, Know Thy Neighbor's Neighbor: The Role of Lookahead in Randomized P2P Networks, *Proc. 36th ACM Symposium on Theory of Computing (STOC 2004)*, p 54–63, June 2004.

We also analyze the Bit-Collection protocol for routing in Sparse-Chord and Sparse-Hypercube. These results have previously appeared in:

[M03] G S MANKU, Routing Networks for Distributed Hash Tables, *Proc. 22nd ACM Symposium on Principles of Distributed Computing (PODC 2003)*, p 133–142, July 2003.

⋆ **Chapter 9 (Mariposa: A Randomized Butterfly)**

We construct Mariposa, a randomized adaptation of butterfly networks. With $3\ell + 3$ out-going links per node, Mariposa can route in $O(\log n/\log \ell)$ hops in the worst-case, which is asymptotically optimal. This constitutes the first randomized network construction to achieve this bound. The construction improves upon Viceroy [MNR02], also an adaptation of butterfly networks, which routes in $O(\log n)$ hops with $\Theta(1)$ out-going links per node. The construction also appears in:

[M03] G S MANKU, Routing Networks for Distributed Hash Tables, *Proc. 22nd ACM Symposium on Principles of Distributed Computing (PODC 2003)*, p 133–142, July 2003.

⋆ **Chapter 10 (Summary)**

We summarize the overall design of Dipsea and present directions for further research.

Chapter 2

Balanced Binary Trees

The trees that are slow to grow bear the best fruit.

MOLIERE (1622–1673)

In this Chapter, we present efficient algorithms for ID Management in Dipsea (see Figure 1.1 on page 2 for a block-diagram of its architecture). The ID Management module is responsible for assigning IDs to new managers – hosts on the Internet – as they join the system. It also re-assigns the IDs of a few existing managers in response to arrivals and departures. At any instant, the current set of IDs divides the hash table into disjoint partitions. Our goal is to devise *decentralized* algorithms that ensure that the variation in partition sizes is minimal, at the cost of as few messages and as few re-assignments of existing IDs as possible.

2.1 Introduction

Consider a dynamic sets of *managers* lying on the circumference of a circle. The managers divide the circumference into disjoint arcs. Each manager manages those points that lie on the arc between itself and its clockwise successor along the circle. Three operations on the set of managers are allowed: (a) a new manager can be inserted into the set, (b) an existing manager can be deleted, and (c) an existing manager can be re-assigned to a new position on the circumference of the circle. The system is *decentralized*. We model decentralization by assuming that no manager has knowledge of other managers at any time. However, their positions can be inferred in two ways:

1. *Random Probe*: The manager of a randomly chosen point on the circumference can be ascertained by paying a cost of R messages.
2. *Local Probe*: The positions of k managers adjacent to a manager can be ascertained by that manager at the cost of $2k$ messages. We call this a "Local Probe of size k".

Our goal is to design message-efficient algorithms for inserting, deleting and re-assigning managers so that the variation in arc sizes is small and the number of re-assignments is minimal. We will quantify the variation by σ, defined as the ratio between the lengths of the longest and the shortest arcs. The following schemes are known:

- No Probes: Upon arrival, a manager is placed at a random point on the circle. No existing manager is re-assigned. The message complexity is zero, since no local or random probes are used. However, $\sigma = \Theta(n \log n)$ (see King and Saia [KS04]).

- One Random Probe: Upon arrival, a manager identifies the location of one existing manager by sending a random probe and *splits* the portion of the circle it manages. After n arrivals, $\sigma = \Theta(\log n)$ (see Naor and Wieder [NW03] or Adler *et al* [AHKV03]). The scheme is known to handle only arrivals, at the cost of R messages per arrival.

- $(c \log n)$ Random Probes: We split the largest of the managers obtained from $(c \log n)$ random probes, where c is a suitably large constant. Naor and Wieder [NW03] and Abraham *et al* [AAA+03] have shown that $\sigma = \Theta(1)$ for this scheme. The cost is $O(R \log n)$ messages per arrival. Karger and Ruhl [KR04] propose an elegant variation on this idea that supports departures of managers as well. However, their variation necessitates $O(\log \log n)$ managers to be re-assigned in response to both arrivals and departures.

- One Random Probe plus One Local Probe of size $(c \log n)$: We carry out one random probe to identify a manager, followed by a local probe of size $c \log n$ in the vicinity of that manager, and split the largest manager encountered. The scheme guarantees $\sigma \leq 4$ at the cost of $O(R + \log n)$ messages [M04]. In response to deletion of a randomly chosen manager, a local probe of size $c \log n$ is carried out in the vicinity of the departed manager and at most one existing manager is re-assigned.

- r Random Probes plus r Local Probes of size v, with $(rv \geq c \log n)$: We carry out r random probes, followed by local probes of size v in the vicinity of the r managers.

We then split the largest manager encountered. The scheme guarantees $\sigma = \Theta(1)$ at the cost of $O(rR+v)$ messages if each node maintains knowledge of v of its neighbors. When $R = o(\log n)$, the scheme is superior to previous schemes. The scheme does not handle departures of managers although a simple heuristic is known to perform well (see reference [KM04] or Chapter 3).

In this Chapter, we explore the "One Random Probe plus One Local Probe of size $(c \log n)$" scheme and some of its variations. The scheme forms the basis of a practical ID management algorithm for Dipsea. The last scheme ("r Random Probes plus r Local Probes of size v, with $(rv \geq c \log n)$") is a generalization of earlier schemes and requires a novel proof technique. This will be the subject of Chapter 3.

Summary of Results

In §2.2, we establish the relationship between managers on a circle and ID Management in Dipsea.

In §2.3, we analyze a stochastic process for growing binary trees corresponding to the "One Random Probe plus One Local Probe of size $(c \log n)$" scheme. The leaf nodes of the resulting tree belong to at most three different levels, with high probability.

In §2.4, we establish the relationship between balanced binary trees and ID Management in Dipsea. The scheme guarantees $\sigma \leq 4$ at the cost of $O(R + \log n)$ messages per arrival.

In §2.5, we modify binary trees slightly by allowing parents of leaves to have high degree. This variation extends the basic idea in §2.3 to guarantee $\sigma \leq 1 + \epsilon$, for any $\epsilon > 0$, albeit at the cost of $O(R + \frac{1}{\epsilon^2} \log n)$ messages and re-assignment of $O(1/\epsilon)$ existing managers per arrival and departure. Ours is the first algorithm that allows such fine-tuning.

In §2.6, we outline a simple variant of the ID Management algorithm in §2.4 that has experimentally been observed to yield $\sigma \leq 4$. We also outline a deletion algorithm that works in conjunction with the addition algorithm. We cannot presently analyze these simple algorithms. Instead, in the next Section, we develop a rather complex algorithm that handles both additions and deletion, and also affords analysis.

In §2.7, we incorporate deletions of randomly chosen managers into our scheme. We guarantee $\sigma \leq 4$ at the cost of $O(R + \log n)$ messages per arrival or departure. At most one existing manager is re-assigned when a manager departs, which is optimal.

In §2.8, we compare our ID Management algorithm with previous proposals for the same.

In §2.9, we summarize and present future research directions.

2.2 ID Management in Distributed Hash Tables

In the context of Dipsea, the circle corresponds to the unit interval $\mathcal{I} = [0, 1)$, with each manager possessing an ID in $\mathcal{I}$. The managers communicate with each other through a *routing network*, a graph structure that is a function of the current set of IDs. The managers are hosts on the Internet and the routing network consists of TCP connections among the hosts. The current set of IDs divides $\mathcal{I}$ into disjoint *partitions*, managed by one host each. In a homogeneous system, we would like to ensure that each participating host is assigned a fair share of the overall load. To quantify the variation in load, we define σ, the *partition balance ratio*, to be the ratio between the largest and smallest partition sizes.

In Dipsea, the routing network consists of two types of links per node: (a) *short-distance* links made with f successors and f predecessors along the circle, where f is a tunable parameter, and (b) a few *long-distance* links with other nodes. The short-distance links constitute a *fault-tolerant ring*. The long-distance links provide short routes among nodes, and will be the subject of Chapters 5 through 9 of this dissertation. In this Chapter, we treat the routing network as a black-box that supports two operations:

1. *Local Probe*: Using the short-distance links, it is possible for a manager to retrieve the IDs of k managers adjacent to a manager in $2k$ messages. In fact, with $f = \Omega(\log n)$, the ring remains intact w.h.p. even if half the managers suddenly die [LNBK02]. Thus in practice, as long as $k \leq f$, the local probe is free of cost.

2. *Random Probe*: Using the long-distance links, it is possible to route a message to the manager of a randomly-chosen point in $\mathcal{I}$ by paying a cost of R messages, with high probability[†]. The earliest DHT routing networks were based on the hypercube and its variants. With n managers, these networks can route in $R = \Theta(\log n)$ messages, with only $\Theta(\log n)$ connections per node. Examples of these networks are Chord [SMK+01, GM04], Pastry [RD01a] and Tapestry [ZHS+04]. Later papers have shown that it is possible to achieve $R = \Theta(\log n / \log\log n)$ with the same number of connections. Examples of these networks are high-degree de Bruijn networks, as has been observed by several groups [AAA+03,FG03,KK03,LKRG03,NW03], high-degree butterflies [KMXY03], Mariposa: a Kleinberg-style randomized butterfly [M03], and several other randomized networks that were analyzed in a recent paper [MNW04] (for

[†] By "with high probability" (w.h.p.), we mean "with probability at least $1 - O(n^{-\lambda})$ for some constant $\lambda > 1$, for a system with n participants".

example, randomized-Chord [ZGG03, GGG$^+$03], randomized-hypercubes [GGG$^+$03], Symphony [MBR03], skip-graphs [AS03] and SkipNet [HJS$^+$03]). We discuss these results in Chapter 8 of this dissertation.

Dipsea consists of a *dynamic* set of managers that join and leave the system frequently. Upon arrival, a new manager has to select an ID for itself[†]. If the current set of IDs could be retrieved from some central location, choosing an ID would be easy. However, Dipsea is *decentralized* — there is no global knowledge of the current set of IDs. IDs of other managers can be inferred by sending local and global probes, as described above.

From a systems standpoint, a good ID management algorithm should enjoy each of the following properties: (a) the algorithm should be decentralized and simple to implement, (b) the algorithm should entail low-cost, measured in terms of messages required for various probes, (c) the variation in partition sizes should be small to ensure load balance among managers, and (d) in response to arrivals and departures, the number of ID re-assignments of existing hosts should be minimal.

Each of the schemes listed in Section 2.1 constitutes an ID management scheme for DHTs. We study the "One Random Probe plus One Local Probe of size $(c \log n)$" scheme in this Chapter. This algorithm is the first to enjoy all of the following properties: (a) both arrivals and departures of hosts are handled, (b) departure of a host causes at most one existing host to change its ID, (c) the ratio of the largest to the smallest partition is at most 4, with high probability, and (d) the expected cost per arrival/departure is $\Theta(R + \log n)$ messages, where n denotes the current number of participants, and R denotes the cost of routing one message using the long-distance links.

Our ID Management algorithm enjoys an additional property that is important: it assumes the existence of short-distance links that support local probes, but is *independent* of the long-distance links, which are treated as a black-box supporting random probes. In contrast, a scheme by Adler *et al* [AHKV03] also guarantees $\sigma = \Theta(1)$ at the cost of $\Theta(\log n)$ messages. However, the scheme has been designed for a specific set of long-distance links: the hypercube. The idea is to identify a manager with one random probe, and to split the largest of this manager's long-distance neighbors, as defined by a hypercubic routing network. The advantage of having an ID management scheme that is independent of the long-distance links is that the two modules: ID Management and Choice of Long-distance

[†]It is customarily assumed in DHT design that a newly-arrived manager "knows" one existing member of the ring at the outset.

Links (see Figure 1.1 on page 2) are cleanly separated.

Variations of our algorithm diminish the ratio between the largest and the smallest partition to $(1+\epsilon)$, for any $\epsilon > 0$, albeit at the cost of re-assigning the IDs of $O(\frac{1}{\epsilon})$ existing hosts per arrival/departure. Ours is the first algorithm that allows such fine-tuning (see §2.5). Finally, our ID management algorithm enables (a) estimation of the total number of hosts in the system by making only local measurements, and (b) emulation of a variety of deterministic and randomized families of routing topologies, in a straightforward fashion. Among these families are several networks that require $O(\log n/\log k)$ routing hops in an n-node network with k links per node. We discuss these features of our ID management algorithms in Chapter 4.

2.3 Three-Level Binary Trees

In this Section, we describe a stochastic process for growing binary trees that guarantees that leaf nodes belong to at most three different levels with high probability. Each internal node in the tree has degree two. The left and right branches of internal nodes are labeled 0 and 1 respectively. The *level* of a leaf node is the length of the path from the root. The root is at level 0. There are at most 2^ℓ leaves at level ℓ.

We mark a small fraction of internal nodes as *active*. Let n denote the number of leaf nodes. With $n = 2$, there is only one internal node, the root node, which is marked active. At all times, we maintain the property that for every leaf node, exactly one internal node along the path from that leaf node to the root is active. Thus the set of active nodes constitutes a *frontier* such that sub-trees hanging below active nodes partition leaf nodes into disjoint groups. We grow the binary tree in a randomized fashion by increasing the number of leaf nodes. Insertion of a new leaf is done in three steps:

A. *Random walk down the tree to reach leaf node* $\mathbf{r}$: Start at the root. At each step, choose between the two children of an internal node, uniformly at random. Let $\mathbf{r}$ denote the leaf node reached.

B. *Perfect insertion in the sub-tree rooted at* $\mathbf{a}$*, the active ancestor of* $\mathbf{r}$: Imagine that each internal node is labeled with the number of leaf nodes in the sub-tree rooted at that node. Starting at $\mathbf{a}$, repeatedly move to the child with fewer leaves below it, breaking ties arbitrarily. Split the leaf node reached, into two.

C. *Check if* $\mathbf{a}$ *should continue to be active*: Let ϕ denote a monotonically non-decreasing function such that $\phi(\ell) \in [0, \ell]$ for non-negative integer ℓ. We maintain the invariant that whenever a new node is inserted at level ℓ under an active node $\mathbf{a}$, then $\mathbf{a}$ is guaranteed to be at level $\phi(\ell)$. Therefore, we mark $\mathbf{a}$ as non-active and we mark both its children as active iff two conditions are satisfied:

1. No more leaves at level ℓ can be created in the sub-tree rooted at $\mathbf{a}$, where ℓ is the level of the leaf just created.
2. $\phi(\ell) \neq \phi(\ell+1)$, where $\phi(\ell)$ is the level of $\mathbf{a}$.

Different choices of ϕ result in a spectrum of algorithms. For example, if $\phi(\ell) = 0$ for all ℓ, we obtain a *perfectly balanced binary tree.* If $\phi(\ell) = \ell$, we obtain a *random binary tree.* We will use the following function in this paper:

$$\phi(\ell) = \max\{0, \ell - \lceil \log_2 \ell \rceil - c\}$$

where c denotes a small constant. The resulting tree is height-balanced, as the following theorem claims:

Theorem 2.1. *With n leaf nodes, all leaves lie in levels $[\log_2 n]$ and $[\log_2 n] \pm 1$, w.h.p., for a suitable choice of c, where $[x]$ denotes the integer closest to real number x.*

To prove Theorem 2.1, we first define the following function:

$$\begin{array}{rcll} \chi(\ell) & = & 1 & \text{if } \ \ell = 0 \\ & & 2^{\ell-1} & \text{if } \ \phi(\ell) = 0 \ \text{ but } \ell \neq 0 \\ & & 2^{\lceil \log_2 \ell \rceil + c - 1} & \text{otherwise} \end{array}$$

We now define

$$Y(\ell) = \sum_{i=0}^{\ell} G_{\chi(i),\phi(i)}$$

where $G_{\chi(i),\phi(i)}$ denotes the sum of $\chi(i)$ geometric random variables, each with probability parameter $2^{-\phi(i)}$. Clearly, $\mathbf{E}Y(\ell) = \sum_{i=0}^{\ell} \chi(i) 2^{-\phi(i)} = 1 + \sum_{i=1}^{\ell} 2^{i-1} = 2^{\ell}$. What is our interest in $Y(\ell)$? Consider $\mathbf{a}$, an active node awaiting insertion of a new leaf at level ℓ. Our algorithm ensures that $\mathbf{a}$ must be at level $\phi(\ell)$. The probability that a newly-arrived host gets inserted as a leaf in the sub-tree rooted at $\mathbf{a}$ is $2^{-\phi(\ell)}$. Thus the total number of node arrivals before a leaf gets inserted below $\mathbf{a}$ happens to be a geometric variable with probability $2^{-\phi(\ell)}$. Now $\chi(\ell)$ equals the number of leaf nodes inserted at level ℓ in the sub-tree rooted at **a** when **a** was active. In the past, each node along the path from the root to **a** has played the role of an active node. Thus the sum $Y(\ell)$ denotes the total number of node arrivals such that all leaf positions at level ℓ in a specific sub-tree rooted at level $\phi(\ell)$ are occupied.

Lemma 2.3.1. *Let $N = 2^\ell$.*
(a) $Pr[Y(\ell) > (1+\delta)N] = O(1/N^2)$, if constant c is chosen suitably (as a function of δ).
(b) $Pr[Y(\ell) < (1-\delta)N] = O(1/N^2)$ if constant c is chosen suitably (as a function of δ).

Proof. A formal proof is presented in Appendix A. The intuition behind the claims is as follows. Consider $G_{\chi(\ell),\phi(\ell)}$, the sum of $\chi(\ell)$ geometric random variables, each with probability parameter $2^{-\phi(\ell)}$. Now $\chi(\ell) = c' \log N$ and $2^{-\phi(\ell)} = 2c' \frac{\log N}{N}$, for some constant c' (which is a function of constant c). Using standard Chernoff bounds, we can claim the following:

(a) $Pr[G_{\chi(\ell),\phi(\ell)} > (1+\delta)(N/2)] < 1/(N/2)^2$, if constant c is chosen suitably (as a function of δ).

(b) $Pr[G_{\chi(\ell),\phi(\ell)} < (1-\delta)(N/2)] < 1/(N/2)^2$ if constant c is chosen suitably (as a function of δ).

In general, $G_{\chi(\ell-i),\phi(\ell-i)}$ is the sum of $\chi(\ell-i)$ geometric random variables, each with probability parameter $2^{-\phi(\ell-i)}$. Now $\chi(\ell-i) = c' \log(N/2^i)$ and $2^{-\phi(\ell-i)} = 2c' \frac{\log(N/2^i)}{N/2^i}$, for some constant c'. The challenge lies in summing these geometric variables for all values of i from 0 through ℓ. The proof in Appendix A derives tail-bounds for such a sum from first principles (by following the technique illustrated in Motwani and Raghavan [MR95]). $\square$

Lemma 2.3.2. *Let the total number of IDs be n.*

(a) If $2^{\ell-1} < n < \frac{7}{8}2^\ell$, then w.h.p., no leaf is at level $\ell+1$ or more (for a suitably large constant c).

(b) If $\frac{5}{4}2^\ell < n < 2^{\ell+1}$, then w.h.p., no leaf is at level $\ell-1$ or less (for a suitably large constant c).

Proof. (a) Let $N = 2^\ell$. Consider a specific leaf node **r**.

$$(\mathbf{r} \text{ is at level } \ell+1 \text{ or more within } n < \tfrac{7}{8}2^\ell \text{ steps}) \quad \Rightarrow \quad Y(\ell) < \tfrac{7}{8}N$$

Plugging $\delta = \frac{1}{8}$ in Lemma 2.3.1(b), we get $Pr[Y(\ell) < (1-\delta)N] < 1/N^2$ for suitably large c. Now, $1/N^2 = O(1/n^2)$. Summing over all n nodes, we arrive at the claim.

(b) Let $N = 2^{\ell}$. Consider a specific leaf node $\mathbf{r}$.

$$(\mathbf{r} \text{ is at level } \ell - 1 \text{ or less even when } n > \tfrac{5}{4}N) \quad \Rightarrow \quad Y(\ell) > \tfrac{5}{4}N$$

Plugging $\delta = \frac{1}{4}$ in Lemma 2.3.1(a), we get $Pr[Y(\ell) > (1+\delta)N] < 1/N^2$ for suitably large c. Now, $1/N^2 = O(1/n^2)$. Summing over all n nodes, we arrive at the claim. □

Theorem 2.1 follows from Lemma 2.3.2. A few notes:

- ⋆ The constants $\frac{7}{8}$ and $\frac{5}{4}$ can be replaced by any r_1 and r_2 satisfying $r_2 < 2r_1$ and $0 < r_1 < 1 < r_2$.
- ⋆ The leaf nodes lie in as many as three different levels only when n hovers around a power of two. Otherwise, leaf nodes belong to only two different levels.
- ⋆ Theorem 2.1 can also be proved using a different proof technique that is presented in Chapter 3, where we develop a generalization of the scheme we studied in this Section.

2.4 An Insert-Only ID Management Algorithm

The relationship between binary trees (described in §2.3) and ID Management in Dipsea is as follows. Only leaf nodes of the tree correspond to IDs. The internal nodes of the tree are conceptual. The sequence of 0s and 1s along the path from the root to a leaf node, treated as the binary expansion of a fraction in $[0, 1)$, constitutes the ID of that leaf.

Step A *(Random walk down the tree)* is equivalent to identifying $\mathbf{r}$, the manager of a point chosen uniformly at random from the interval $\mathcal{I} = [0, 1)$.

Step B (*Perfect insertion into the sub-tree rooted at* $\mathbf{a}$*, the active ancestor of* $\mathbf{r}$) can be accomplished as follows: Let $\mathbf{r}$ have an ℓ-bit ID. Let $S(\mathbf{r})$ denote the set of IDs that share the top $\phi(\ell)$ bits with $\mathbf{r}$. If $|S(\mathbf{r})| \geq 2^{\ell-\phi(\ell)}$, then we split $\mathbf{r}$. Otherwise, there exists an $(\ell-1)$-bit ID $\mathbf{s} \in S(\mathbf{r})$, which we split. *Splitting* a node $\mathbf{x}$ amounts to replacing $\mathbf{x}$ with two IDs: $\mathbf{x}0$ and $\mathbf{x}1$. Treated as fractions in $[0, 1)$, $\mathbf{x}$ and $\mathbf{x}0$ are equivalent. The newly arrived host is assigned $\mathbf{x}1$.

Step C (*Check if* $\mathbf{a}$ *should continue to be active*) is implicit in step B above.

Theorem 2.2. *A newly-arrived host needs $\Theta(R + \log n)$ messages w.h.p. to obtain an ID, where n denotes the current number of hosts. The partition balance ratio is $\sigma \leq 4$.*

Proof. The cost of identifying $\mathbf{r}$, the manager of a random number in $[0, 1)$ costs R messages, by definition of R. Computing $S(\mathbf{r})$ entails $\Theta(\log n)$ messages if we use successor/predecessor links along the circle. This is because $|S(\mathbf{r})| \leq 2 \times 2^{\ell-\phi(\ell)} = \Theta(\ell)$ (from the definition of $\phi(\ell)$) and $\ell = \Theta(\log n)$ (Theorem 2.1). A host with an ℓ-bit ID manages a sub-interval of size $2^{-\ell}$. From Theorem 2.1, the leaves are in at most three different levels. Therefore, $\sigma \leq 4$. $\square$

Optimization: In a practical system, $S(\mathbf{r})$ could be maintained as the value associated with a hash key that includes $\mathbf{a}$, the active ancestor of $\mathbf{r}$, as a sub-key. Alternately, a specific node within the sub-tree rooted at $\mathbf{a}$ (for example, the leftmost child of the sub-tree) could be made responsible for this data structure. In either case, the data structure can be retrieved and updated in $\Theta(R)$ messages.

2.5 High-Degree Trees for Fine-grained Partition Balance

We can guarantee that the partition balance ratio satisfy $\sigma \leq 1 + \epsilon$, for any $\epsilon > 0$, if we modify the tree in §2.3 a little: we stipulate that the degree of parents-of-leaves (internal nodes that have leaf nodes as their children), could have any value in the range $[b, 2b-1]$ for some $b \geq 2$. All other internal nodes still have degree exactly two. The smallest trees we consider have b leaf nodes. The degree constraint is reminiscent of B-trees [BM72] but limited to parents-of-leaves.

What is the ID associated with a leaf node $\mathbf{x}$ at level ℓ? Let $p \in [0, 1)$ be the ID associated with the parent of $\mathbf{x}$. If $\mathbf{x}$ is the i^{th} out of a total of c children of its parent, its ID is $p + \frac{i-1}{c}2^{-\ell}$.

Steps A and B in §2.3 remain the same. Step C is different due to two changes. First, we use the function $\phi(\ell) = \ell - \lceil \log_2 \ell \rceil - c' \log \epsilon^{-2}$ for a suitably large constant c'. Second, the properties associated with an active node are different: when an internal node $\mathbf{x}$ becomes active for the first time, the degree of parents-of-leaves within the sub-tree rooted at $\mathbf{x}$, is exactly b. None of these nodes becomes degree $b+2$ unless all of them are degree $b+1$, and so on. Finally, enough leaves would have arrived so that all these parents-of-leaves below $\mathbf{x}$ have degree $2b-1$. Thereafter, insertions below $\mathbf{x}$ split degree-$(2b-1)$ parents-of-leaves into pairs of internal nodes, each with degree b.

Every insertion changes the degree of some internal node. Each such change is concomitant with re-assignments – at least $b-1$ and at most $2b-2$ existing hosts get re-assigned.

We now establish that by setting $b = 2/\epsilon$, and by choosing a suitably large constant c', the tree is highly balanced. At all times, there at most three different (level, degree) pairs corresponding to parents-of-leaf nodes, w.h.p. For $b \geq 2$, these pairs are

$$\begin{aligned} & (\ell, d), (\ell, d+1), (\ell, d+2), \quad \text{where } d \in [b, 2b-3] \\ \text{or} \quad & (\ell, 2b-2), (\ell, 2b-1), (\ell+1, b) \\ \text{or} \quad & (\ell, 2b-1), (\ell+1, b), (\ell+1, b+1) \end{aligned}$$

For $b = 2$, these three pairs are

$$\begin{aligned} & (\ell, 2), (\ell, 3), (\ell+1, 2) \\ \text{or} \quad & (\ell, 3), (\ell+1, 2), (\ell+1, 3) \end{aligned}$$

As a consequence, $\sigma \leq 1 + 2/b$ w.h.p. Since $b = 2/\epsilon$, we get $\sigma \leq 1 + \epsilon$.

Theorem 2.3. *The modified ID selection scheme guarantees $\sigma \leq 1 + \epsilon$, w.h.p., when $\phi(\ell) = \ell - \lceil \log_2 \ell \rceil - c' \log \epsilon^{-2}$ for a suitably chosen constant c'.*

Proof. We define operators $\preceq$ and $\succeq$ as follows:

$$\begin{aligned} (x, y) \preceq (x', y') \quad & \text{iff} \quad (x < x') \text{ or } ((x = x') \text{ and } (y \leq y')). \\ (x, y) \succeq (x', y') \quad & \text{iff} \quad (x > x') \text{ or } ((x = x') \text{ and } (y \geq y')). \end{aligned}$$

Using arguments similar to those in Lemma 2.3.1, we can show that for a suitably large value of c', if $\frac{5}{4}(b+i)2^\ell < n < (b+i+1)2^\ell$ for $0 \leq i < b$, then w.h.p., $(\ell+1, b+i) \preceq$ (level, degree) pair for any parent-of-leaves. Similarly, we can show that (a) if $(b+i-1)2^\ell < n < \frac{7}{8}(b+i)2^\ell$, for $1 \leq i < b$, then w.h.p., $(\ell+1, b+i) \succeq$ (level, degree) pair for any parent-of-leaves, and (b) if $(2b-1)2^\ell < n < \frac{7}{8}(2b)2^\ell$, then w.h.p., $(\ell+2, b) \succeq$ (level, degree) pair for any parent-of-leaves. It follows that $\sigma \leq 1 + 2/b$ w.h.p. Since $b = 2/\epsilon$, we get $\sigma \leq 1 + \epsilon$. □

Notes: In terms of levels, for $b \geq 2$, leaves belong to two different levels only when n is very close to $b2^\ell$ for some ℓ; otherwise, leaves occupy just one level.

2.6 Practical ID Management

A simple variant of the ID Management algorithm developed in Section 2.4 works as follows:

> *Identify* $\mathbf{r}$*, the manager of a random number in* $[0, 1)$*. Let* $\mathbf{r}$ *have an* ℓ*-bit ID. Identify the IDs of* $c\ell$ *managers in the vicinity of* $\mathbf{r}$ *along the circle, where* c *is a suitably-large constant. Split the largest manager into two.*

The resulting set of IDs still corresponds to leaf nodes of a binary tree. Experiments indicate that the scheme results in $\sigma \leq 4$ w.h.p. The primary difference from the algorithm in Section 2.4 is that the "vicinity" of a host has not been restricted to be a proper subtree. It would be an interesting exercise to adapt the proof of Theorem 2.1 to this general strategy of identifying the manager to split.

A simple deletion algorithm that works in conjunction with the above scheme for insertion and has experimentally been observed to guarantees $\sigma \leq 4$, works as follows:

> *Let* $\mathbf{r}$*, a randomly-chosen manager that departs, have an* ℓ*-bit ID. Identify the IDs of* $c\ell$ *managers in the vicinity of* $\mathbf{r}$ *along the circle, where* c *is a suitably-large constant. Re-assign the smallest manager to occupy the position of the manager that departed, unless such a re-assignment increases the size of the largest manager in the vicinity.*

It would be interesting to devise proof techniques to analyze the above algorithms since they are quite simple. In the next Section, we develop a rather complex scheme that handles both additions and deletions. IDs correspond to leaf nodes of a binary tree. We guarantee $\sigma \leq 4$ by ensuring that leaf nodes lie in at most three different levels.

2.7 Handling Host Departures

In this Section, we develop a variant of our algorithm in §2.3 to handle host departures as well. From a practical perspective, the algorithm briefly described in §2.6 handles both insertions and deletions while ensuring $\sigma \leq 4$. From a theoretical perspective, it would be nice to devise a proof for the scheme in §2.6 since the scheme we develop in this Section entails considerable book-keeping.

The intuition underlying our scheme in this Section is as follows: if n hosts choose random numbers in $[0, 1)$ uniformly at random, then an interval of size $\frac{c \log n}{n}$ receives at least $\frac{1}{2} c \log n$ and at most $2c \log n$ hosts with high probability, if c is a sufficiently large constant. Imagine that we divide $[0, 1)$ into disjoint sub-intervals of size 2^{-i} each, where $i = \left\lceil \log_2 \frac{n}{c \log n} \right\rceil$. Now, if we could *perturb* the random numbers chosen by all the hosts such that each sub-interval of size 2^{-i} is (almost) evenly divided among those hosts whose random numbers fall into that sub-interval, we could guarantee $\sigma \leq 4$. The challenge lies (a) in making sure that at most one existing ID is re-assigned per arrival/departure, and (b) in decentralized maintenance of sub-interval boundaries as n itself changes over time.

2.7.1 Overview

We create binary trees, as before. The ID of a leaf node is the sequence of 0s and 1s from the root to that node. When a randomly-chosen host departs, we restore balance in the tree by making an adjustment in a small-sized sub-tree that includes the departed host. As before, a "frontier" of internal nodes defines the set of sub-trees within which local adjustments are made. The challenge lies in maintaining the frontier in response to both arrivals and departures, and being able to analyze the shape of the resulting tree (in terms of levels to which various leaf nodes belong).

We maintain a 1-1 correspondence between hosts and leaf nodes in the tree, their IDs being the same. When a host joins, it arrives with an infinite string of random bits. During the lifetime of a host, it can swap its bit-string with that of another host. Each host possesses exactly one bit-string at any time – this bit-string is not necessarily the one it joined the system with. The departure of a randomly chosen host is equivalent to deletion of a randomly chosen bit-string in the system. Swapping of bit-strings is different from *re-assignment* of host IDs, which occurs only when the tree itself undergoes structural changes.

In the next Section, we describe two simple procedures on sub-trees.

2.7.2 Perfect Insertion and Perfect Deletion

Definition of "perfect insertion of a newly-arrived host below internal node $\mathbf{x}$": Starting at $\mathbf{x}$, we repeatedly move down either the left or the right sub-tree, whichever has fewer leaf nodes, breaking ties arbitrarily. Let $\mathbf{r}$ denote the leaf node we reach. We *split* $\mathbf{r}$ into two by creating two leaf nodes. The newly-arrived host occupies the right leaf. The host that occupied $\mathbf{r}$ now occupies the left leaf.

Definition of "perfect deletion of a specific host (corresponding to some leaf node) below a specific internal node $\mathbf{x}$": Let the host correspond to leaf node $\mathbf{r}$ at level ℓ. If ℓ is the deepest level at which leaf nodes exist below $\mathbf{x}$, we delete both $\mathbf{r}$ and its sibling from the tree. The host at $\mathbf{r}$ leaves the system. The host at the sibling effectively "moves up", its ID possibly having been re-assigned since we chopped off its least-significant bit. If ℓ is not the deepest level below $\mathbf{x}$, we identify some pair of sibling leaf nodes that lie at the deepest level. We delete both of these leaf nodes. In terms of hosts, the host corresponding to the left sibling "moves up", its ID remaining effectively the same (we chopped off its least-significant bit, which was 0). The host corresponding to the right sibling is re-assigned – it takes up the ID corresponding to leaf node $\mathbf{r}$. The host that previously occupied $\mathbf{r}$ leaves the system. The bit-string in its possession also vanishes from the system.

2.7.3 Definitions

Each internal node belongs to one of four states: B, F, F* or A. The letters are acronyms for the words BELOW, FRONTIER, FRONTIER* and ABOVE, respectively. We maintain the invariant that for every leaf node, exactly one of its ancestors is marked as F or F* at any time. Thus the union of F and F* nodes acts as a frontier, partitioning leaf nodes into disjoint groups. Every internal node that is *above* the frontier is in state A. All other internal nodes are *below* and in state B. The frontier "moves down" in response to additions and "moves up" in response to deletions. When a new host arrives, the state of an existing node either remains the same, or one of the following transitions takes place: B→F, B→F*, F→F*, or F*→A. In response to deletions, either the state remains the same, or one of the following transitions occurs: B←F, B←F*, F←F*, or F*←A.

For brevity, we will use $\mathbf{x}$ to denote both an internal node and the string of 0s and 1s from the root to that node. Thus $\mathbf{x}0$ and $\mathbf{x}1$ denote the left and the right child of $\mathbf{x}$, respectively.

Definitions: Function $N(\mathbf{x})$ denotes the number of random bit-strings with prefix $\mathbf{x}$, currently in the system. Function ϕ remains unchanged: $\phi(\ell) = \max\{0, \ell - \lceil \log_2 \ell \rceil - c\}$. We define function ψ for any integer $k \geq 0$:

$$\psi(k) = 2^{x-k} \quad \text{where } x = \max\{\ell \mid \phi(\ell) = k\}$$

Notes:

If $n = 2^k$, then $\psi(\phi(k)) \geq d \log n$ for some constant d (that depends upon constant c).
From the definition of ϕ, we infer that $\psi(k+1)$ equals either $\psi(k)$ or $2\psi(k)$, for all $k \geq 0$.
For string $\mathbf{x}$, $\psi(\mathbf{x}) \equiv \psi(|\mathbf{x}|)$, where $|\mathbf{x}|$ denotes the length of string $\mathbf{x}$.

Invariants: We maintain the following invariant for every node $\mathbf{x}$ in state A, F or F*: *a bit-string is possessed by some leaf node below* $\mathbf{x}$ *iff* $\mathbf{x}$ *is a prefix of that bit-string.* A node in state B (the "non-frontier" nodes) may or may not satisfy this invariant. Figure 2.1 describes six additional invariants that are always maintained. The addition and deletion algorithms in §2.7.4 and §2.7.5 basically show how all the invariants can be maintained in response to arrivals and departures of hosts.

$$\begin{array}{llccccc}
\mathbf{x} \text{ is in state F} & \Rightarrow & N(\mathbf{x}) \geq \psi(\mathbf{x}) & \wedge & \big(N(\mathbf{x}0) < \psi(\mathbf{x})/2 & \vee & N(\mathbf{x}1) < \psi(\mathbf{x})/2\big) \\
\mathbf{x} \text{ is in state F}^* & \Rightarrow & N(\mathbf{x}) \geq \psi(\mathbf{x}) & \wedge & \psi(\mathbf{x})/2 \leq N(\mathbf{x}0) < \psi(\mathbf{x}0) & \wedge & \psi(\mathbf{x})/2 \leq N(\mathbf{x}1) < \psi(\mathbf{x}1) \\
\mathbf{x} \text{ is in state A} & \Rightarrow & N(\mathbf{x}) \geq \psi(\mathbf{x}) & \wedge & N(\mathbf{x}0) \geq \psi(\mathbf{x}0) & \wedge & N(\mathbf{x}1) \geq \psi(\mathbf{x}1)
\end{array}$$

Figure 2.1: *Three of the invariants maintained by our algorithm are listed above. For each invariant, the condition on the Right-Hand-Side* AND *the condition that all ancestors of* $\mathbf{x}$ *are in state* A*, implies the Left-Hand-Side. This results in three additional invariants.*

2.7.4 Addition Algorithm

If there are fewer than $\psi(0)$ hosts in the system, all internal nodes are in state B – we maintain a complete binary tree (using "perfect addition" and "perfect deletion" below the root node). When there are exactly $\psi(0)$ hosts in the system, the root node undergoes the transition B→F, or B→F*. If $N(\mathbf{0}) = N(\mathbf{1}) = \psi(0)/2$, then B→F* occurs; otherwise, B→F occurs. For the rest of the Section, we assume that the root is not in state B.

When a new host arrives, it is in possession of an infinite random bit-string. Corresponding to the bits of this bit-string, we carry out a random walk down the tree. Let $\mathbf{r}$ denote the leaf node we reach. Let $\mathbf{a}$ denote the ancestor of $\mathbf{r}$ in state F or F*.

Case I ($\mathbf{a}$ is in state F)

> If $N(\mathbf{a}0) < \psi(\mathbf{a})/2$, we carry out "perfect insertion" below the sub-tree rooted at $\mathbf{a}1$. Otherwise, $N(\mathbf{a}1) < \psi(\mathbf{a})/2$ and we carry out "perfect insertion" below the sub-tree rooted at $\mathbf{a}0$. The bit-string of the newly-arrived host increments either $N(\mathbf{a}0)$ or $N(\mathbf{a}1)$. After the increment, if $\min\{N(\mathbf{a}0), N(\mathbf{a}1)\} \geq \psi(\mathbf{a})/2$, then $\mathbf{a}$ undergoes the transition F→F*; otherwise it remains in state F. Note that in state F*, the number of leaf nodes below $\mathbf{a}0$ and $\mathbf{a}1$ are *exactly* $N(\mathbf{a}0)$ and $N(\mathbf{a}1)$, respectively. This property is maintained in state F*, as described below.

Case II ($\mathbf{a}$ is in state F*)

> If $\mathbf{a}0$ is a prefix of the random bit-string of the newly-arrived host, we carry out "perfect insertion" below $\mathbf{a}0$. Otherwise, we carry out "perfect insertion" below $\mathbf{a}1$. After the insertion, if $N(\mathbf{a}0) \geq \psi(\mathbf{a}0)$ and $N(\mathbf{a}1) \geq \psi(\mathbf{a}1)$, we invoke procedure HANDLE($\mathbf{a}$), described below: HANDLE($\mathbf{x}$) first re-distributes the bit-strings currently in possession of leaf nodes below $\mathbf{x}$: All bit-strings with prefix $\mathbf{x}0$ are made to lie below $\mathbf{x}0$, and all bit-strings with prefix $\mathbf{x}1$ are made to lie below $\mathbf{x}1$. The state of

x then changes to A. At this point, if $N(\mathbf{x}00) \geq \psi(\mathbf{x}00)$ and $N(\mathbf{x}01) \geq \psi(\mathbf{x}01)$, we invoke HANDLE($\mathbf{x}0$) recursively; else if $N(\mathbf{x}00) \geq \psi(\mathbf{x}0)/2$ and $N(\mathbf{x}01) \geq \psi(\mathbf{x}0)/2$, $\mathbf{x}0$ is assigned state F*; else $\mathbf{x}0$ is assigned state F. Node $\mathbf{x}1$ is dealt with similarly. We note that a recursive call is in fact, a rare event, as shown in §2.7.6.

The addition algorithm differs from that in §2.3 in the following way: the transition F→A has been *delayed* by introducing an intermediate state F*. Plus, we maintain some random bit-strings with hosts.

2.7.5 Deletion Algorithm

Let **r** denote a randomly-chosen host that departs. Let **a** denote the ancestor of **r** in state F or F*. Departure of **r** will decrement $N(\mathbf{a})$. Moreover, either $N(\mathbf{a}0)$ or $N(\mathbf{a}1)$ will also be decremented, depending upon the prefix of the random bit-string in possession of **r**. In the cases below, the values of $N(\mathbf{a})$, $N(\mathbf{a}0)$ and $N(\mathbf{a}1)$ correspond to the *updated* values (following the decrement).

Case I: $N(\mathbf{a}) \geq \psi(\mathbf{a})$

a) $\psi(\mathbf{a})/2 \leq N(\mathbf{a}0) < \psi(\mathbf{a}0)$ and $\psi(\mathbf{a})/2 \leq N(\mathbf{a}1) < \psi(\mathbf{a}1)$

We carry out "perfect deletion" of **r** below $\mathbf{a}0$ or $\mathbf{a}1$, whichever happens to be the ancestor of **r**. Node **a** remains in state F*.

b) $N(\mathbf{a}0) < \psi(\mathbf{a})/2$ or $N(\mathbf{a}1) < \psi(\mathbf{a})/2$

We carry out "perfect deletion" of **r** below **a**. Node **a** undergoes F←F* if its state was F* before the departure of **r**.

Case II: $N(\mathbf{a}) < \psi(\mathbf{a})$

We carry out "perfect deletion" of **r** below **a**. Internal node $\mathbf{a}'$, the parent of **a**, undergoes the transition F*←A. Node **a**, its *sibling*, and all descendants of its sibling which are presently in state F or F*, undergo B←F or B←F*, whichever is applicable. This cascade of state-transitions is the "inverse" of the recursive call to procedure HANDLE that was described earlier. Only the sibling changes state w.h.p.

2.7.6 Analysis

Lemma 2.7.1. *If $\frac{5}{4}2^{\ell} < n < 2^{\ell+1}$, then w.h.p., no leaf is at level $\ell-1$ or less (for a suitably large constant c).*

Proof. If $\phi(\ell) \neq \phi(\ell+1)$, the lemma follows from the following series of claims, each holding w.h.p.

A1: If $|\mathbf{a}| \leq \phi(\ell)$, then $N(\mathbf{a}) \geq \psi(\mathbf{a})$.

A2: If $|\mathbf{a}| < \phi(\ell)$, then $\mathbf{a}$ is in state A.

A3: If $|\mathbf{a}| = \phi(\ell)$, then $\mathbf{a}$ is in state A, F or F*.

A4: No leaf is at level $\ell - 1$ or less.

If $\phi(\ell) = \phi(\ell+1)$, the lemma follows from the following series of claims, each holding w.h.p.

B1: If $|\mathbf{a}| \leq \phi(\ell) - 1$, then $N(\mathbf{a}) \geq \psi(\mathbf{a})$.

B2: If $|\mathbf{a}| < \phi(\ell) - 1$, then $\mathbf{a}$ is in state A.

B3: If $|\mathbf{a}| = \phi(\ell) - 1$, then $\mathbf{a}$ is in state A, F or F*.

B4: Let $\mathbf{a}'$ denote the sibling of $\mathbf{a}$. Then $N(\mathbf{a}) \geq \psi(\mathbf{a})/2$ and $N(\mathbf{a}') \geq \psi(\mathbf{a}')/2$.

B5: No leaf is at level $\ell - 1$ or less.

Claim A1 is proved as follows. Let $\eta = 2^\ell$. Then $N(\mathbf{a}) = B(\frac{5}{4}\eta, 2^{-|\mathbf{a}|})$, the sum of $\frac{5}{4}\eta$ Bernoulli variables, each with probability $2^{-|\mathbf{a}|}$. The expected value of $N(\mathbf{a})$ is $\frac{5}{4}2^{\ell-|\mathbf{a}|} \geq \frac{5}{4}\psi(\mathbf{a})$. Since $\psi(\mathbf{a}) \geq d \log \eta$, where d is some constant that depends on c. Application of Chernoff bound shows that the event $N(\mathbf{a}) < \psi(\mathbf{a})$ holds with probability at most $1/\eta^2$ (for a suitably large constant c). A union bound on all n nodes makes the probability of failure at most $O(1/\eta) = O(1/n)$.

Claims A2 and A3 follow from A1 and the invariants in Figure 2.1. Claim A4 follows from the fact that all nodes in level $\ell - 1$ and less are occupied if all nodes in level $\phi(\ell)$ are in state A, F or F*.

Claims B1—B5 can be proved along the same lines. ◻

Lemma 2.7.2. *If $2^{\ell-1} < n < \frac{7}{8}2^\ell$, then w.h.p., no leaf is at level $\ell+1$ or more (for a suitably large constant c).*

Proof. Consider internal node $\mathbf{a}$ with $|\mathbf{a}| = \phi(\ell)$. The lemma follows from the following claims:

C1: If $\phi(\ell) \neq \phi(\ell+1)$, then $N(\mathbf{a}) < \psi(\mathbf{a})$ w.h.p.

C2: If $\phi(\ell) = \phi(\ell+1)$, then $N(\mathbf{a}) < \psi(\mathbf{a})/2$ w.h.p.

C3: In either case, the "next insertion" below any F/F* node is going to be at level ℓ or less. Thus, there is no leaf at level $\ell+1$ or above, w.h.p. ◻

Theorem 2.4. *With n hosts, all leaves lie in levels $[\log_2 n]$ and $[\log_2 n] \pm 1$, where $[x]$ denotes the integer closest to real number x. Thus $\sigma \leq 4$.*

Proof. Follows from Lemmas 2.7.1 and 2.7.2. ☐

2.7.7 Extensions

The algorithm outlined in §2.7.4 and §2.7.5 can be extended along the lines of §2.5 (high-degree trees) to achieve partition balance ratio $1 + \epsilon$, w.h.p. Further, an accurate estimate of n can be obtained by scaling the size of the sub-tree below any internal node in state F or F*. This enables emulation of a variety of inter-connection topologies with only two sets of links per host, as described in Chapter 4.

2.8 Comparison with Previous Work

Early DHT designs allowed each host to independently choose a number in $[0, 1)$ uniformly at random [SMK+01, ZHS+04, RD01a, RFHK01, FG03, KK03, MBR03, MNR02, HJS+03]. This corresponds to the "No Probes" scheme described in §2.1. King and Saia [KS04] recently established that $\sigma = \Theta(n \log n)$ w.h.p., where σ denotes the ratio between the largest to the smallest partition sizes.

If a newly-arrived host first chooses a random number in $[0, 1)$ and then *splits* the partition the number falls into, σ diminishes to $\Theta(\log n)$ (see Adler *et al* [AHKV03] or Naor and Wieder [NW03]). This corresponds to the "One Random Probe" scheme described in §2.1. Further improvement is possible. If each host creates $\Omega(\log n)$ *virtual IDs* [DKK+01], σ reduces to $O(1)$. However, the number of overlay connections per host gets amplified by a factor of $\Omega(\log n)$ – this is costly because higher degree overlay networks require more resources for maintenance.

Two different approaches for ID management have recently been proposed, each of which guarantees $\sigma = \Theta(1)$ with only one ID per host. The first approach [NW03, AAA+03, KR04] is overlay-independent while the second is overlay-dependent [AHKV03]. Naor and Wieder [NW03] and Abraham *et al* [AAA+03] proposed that a new host should choose $\Theta(\log n)$ random points from $[0, 1)$, identify the managers of these points and split the largest manager into two. This corresponds to the "($c \log n$) Random Probes" scheme in §2.1. Karger and Ruhl [KR04] have proposed an elegant variation on the idea that supports departures as well. However, their variation necessitates $O(\log \log n)$ hosts to change their

Algorithm	*Overlay Indep.*	σ	*Message Cost*	*Handles Deletions*	*Number of Re-assigns*
Random Binary Tree	Yes	$\Theta(\log n)$	R	No	–
Adler *et al* [AHKV03]	No	$\Theta(1)$	$\Theta(R + \log n)$	No	–
Naor and Wieder [NW03]	Yes	$\Theta(1)$	$\Theta(R \log n)$	(Yes)	(?)
Abraham *et al* [AAA+03]	Yes	$\Theta(1)$	$\Theta(R \log n)$	No	–
Karger and Ruhl [KR04]	Yes	$\Theta(1)$	$\Theta(R \log n)$	Yes	$O(\log \log n)$
Algorithms in Ch 2	**Yes**	**4**	$\Theta(R + \log n)$	**Yes**	**1**
	Yes	**2**	$\Theta(R + \log n)$	**Yes**	**2**
	Yes	$(1 + \epsilon)$	$\Theta(R + \frac{1}{\epsilon^2} \log n)$	**Yes**	$O(1/\epsilon)$
Algorithms in Ch 3	**Yes**	$\Theta(1)$	$\Theta(rR + v)$ with $rv = \Theta(\log n)$	**No**	-

Table 2.1: *R denotes the average number of messages required by the overlay routing layer. Typically, $R = \Theta(\log n)$ or $R = \Theta(\log n / \log \log n)$, w.h.p. σ denotes the ratio of the largest partition to the smallest partition. For the scheme by Adler et al [AHKV03], $R = \Theta(\log n)$ since the scheme is tied to a specific overlay routing topology: the hypercube.*

IDs in response to both arrivals and departures – a costly operation. Adler *et al* [AHKV03] analyzed an overlay-dependent scheme that is specific to hypercubes. The idea is to identify the manager of a random point in $[0, 1)$, probe other managers it has established overlay connections with, and to split the largest of these managers into two. A scheme for handling departures exists, but it has not yielded to formal analysis yet. The idea in [AHKV03] had earlier been proposed as a heuristic in [RFHK01].

Our algorithm enjoys the following features (see Table 2.1):

i) Generality: Our algorithm is independent of the overlay routing topology.

ii) Low cost: Our algorithm requires $\Theta(R + \log n)$ messages. Other overlay-independent schemes require $\Theta(R \log n)$ messages [AAA+03, NW03, KR04].

iii) Host departures: [AAA+03] and [AHKV03] do not handle departures. A full exposition of the departure-scheme in [NW03] has been deferred to the final version of their paper. The scheme in [KR04] requires $O(\log \log n)$ re-assignments for both arrivals and departures. Departures entail only one re-assignment in our algorithm, for $\sigma \leq 4$.

iv) Optimal re-assignments: To guarantee $\sigma \leq 4$ w.h.p., arrivals engender no ID re-assignments and departures entail re-assignment of at most one existing host. To guarantee $\sigma \leq 1 + \epsilon$, for $0 < \epsilon \leq 1$, only $O(1/\epsilon)$ existing hosts have to be re-assigned.

v) Small partition balance ratios: Ours is the first algorithm that allows σ to be fine-tuned to $1 + \epsilon$, albeit at the cost of $O(1/\epsilon)$ re-assignments per arrival and departure. None of the existing schemes with a single ID per host can obtain $\sigma < 4$.

vi) Spectrum of schemes: A simple generalization of our algorithm yields a variety of schemes ranging from completely centralized (perfectly balanced binary trees) to completely decentralized (random binary trees).

vii) Overlay construction: Our algorithm enables *emulation* of various families of deterministic and randomized inter-connection network topologies. A host makes exactly *three sets* of links with other hosts. See Chapter 4 for a description of the Emulation Engine of Dipsea.

viii) Estimation of n: Our algorithm permits a simple scheme for estimating n, the total number of participants currently in the system. In fact, a factor-4 estimate can be derived from a host's own ID. Sharper estimates can be obtained by modifying the arrivals/departure protocols at no extra cost. The resulting estimation scheme diminishes the number of sets of links required for emulation to only two. See Chapter 4 for further details.

2.9 Summary and Future Directions

We described efficient algorithms for the ID Management layer of Dipsea. Our algorithms ensure that the ratio of the largest to the smallest partition is small. The key idea is for a newly arrived host to identify the manager of a random number in $[0, 1)$, to ascertain the IDs of $(c \log n)$ managers in its vicinity along the circle, and to split the largest manager into two. Departures of hosts are handled similarly: a local adjustment within a neighborhood of $(c \log n)$ managers is made. At most one existing host is re-assigned its ID, which is optimal. We also presented a variation of the basic algorithm that diminishes the ratio of the largest to the smallest partition to $1 + \epsilon$, albeit at the cost of re-assigning the IDs of $O(1/\epsilon)$ existing hosts per arrival and departure.

Possible directions for future work:

1. Experiments indicate that the following scheme for handling arrivals also ensures $\sigma \leq 4$:

   > *Identify* $\mathbf{r}$*, the manager of a random number in* $[0, 1)$*. Let* $\mathbf{r}$ *have an* ℓ*-bit ID. Identify the IDs of* $c\ell$ *managers in the vicinity of* $\mathbf{r}$ *along the circle, where* c *is a suitably-large constant. Split the largest manager into two.*

   The scheme works even if the "vicinity" of a host is not a proper sub-tree. It would be interesting to adapt the proof of Theorem 2.1 to this general strategy of identifying the manager to split.

2. Experiments also indicate that a deletion scheme along the following lines, in conjunction with the insertion scheme outlined above, ensures $\sigma \leq 4$:

   > *Let* $\mathbf{r}$*, a randomly-chosen manager that departs, have an* ℓ*-bit ID. Identify the IDs of* $c\ell$ *managers in the vicinity of* $\mathbf{r}$ *along the circle, where* c *is a suitably-large constant. Re-assign the smallest manager to occupy the position of the manager that departed, unless such a re-assignment increases the size of the largest manager in the vicinity.*

   It would be interesting to devise proof techniques that handle this deletion scheme since the algorithm developed in Section 2.7 is rather complex.

3. Are $\Omega(1/\epsilon)$ re-assignments necessary to guarantee $\sigma = 1 + \epsilon$?

4. ID Management for heterogeneous managers is an interesting open problem. See Godfrey and Stoica [GS04] for initial results along this direction.

Chapter 3

Coupon Collection over Cliques

In this Chapter, we develop an ID Management algorithm for Dipsea (see Figure 1.1 on page 2 for a block-diagram of its architecture). The algorithm is a generalization of the algorithm we developed in Chapter 2, and is superior to the earlier algorithm for certain overlay routing networks.

ID Management is responsible for assigning IDs to new hosts as they join the system. It also re-assigns the IDs of a few existing hosts in response to arrivals and departures. At any instant, the current set of IDs divides the hash table into disjoint partitions. Our goal is to devise *decentralized* algorithms that ensure that the variation in partition sizes is minimal, at the cost of as few messages and as few re-assignments of existing IDs as possible.

In Chapter 2, we devised the following ID Management algorithm:

> One Random Probe plus One Local Probe of size $(c \log n)$: An ID for a newly-arrived manager is derived as follows. We identify the manager of a random number in $[0, 1)$, ascertains the IDs of $(c \log n)$ managers in its vicinity along the circle, and split the partition owned by the largest manager into two equal halves. In response to deletion of a randomly chosen manager, a local probe of size $c \log n$ is carried out in the vicinity of the departed manager and at most one existing manager is re-assigned.

In this Chapter, we develop the following ID Management algorithm:

> r Random Probes plus r Local Probes of size v, with $(rv \geq c \log n)$: A newly arrived host identifies the manager of r random numbers in $[0, 1)$, ascertains the IDs of v managers in the vicinity of each of the r managers, and splits the largest manager into two equal halves.

We show that as long as $rv \geq c \log n$, for a suitably large constant c, the ratio between the largest and the smallest partition is $\Theta(1)$ with high probability. The new algorithm requires $O(rR + v)$ messages per arrival, where R is the average cost of sending a message using the routing network of Dipsea. Apart from being an interesting randomized system for analysis, the new algorithm is superior to the algorithm developed in Chapter 2 when $R = o(\log n)$, which is true for several routing networks.

The algorithm is a generalization of the algorithm in Chapter 2, but handles only host arrivals. A simple heuristic for deletions is experimentally shown to perform well – we do not have a technique for analyzing it yet.

3.1 Introduction

In the standard coupon collector process, there are n types of coupons and in each trial, a coupon is chosen independently and uniformly at random. It is well known that the number of trials needed to collect at least one copy of each type is *sharply concentrated* around $n \log n$ (see [MR95], for example). One generalization is to have multiple choices: in each trial, pick d coupons at random and if any of them is not collected, collect a random uncollected coupon. Another generalization is to introduce a graph structure: coupon collection is carried out on a graph whose nodes correspond to coupons and are initially uncovered. In each trial, pick a node at random and if any of its neighbors is uncovered, cover a random uncovered neighbor. Adler *et al* [AHKV03] call this process *Structured Coupon Collection* over graphs. They establish that with high probability†, $O(n)$ steps suffice to cover all nodes of hypercubes on n nodes, Δ-regular graphs with $\Delta = \Omega(\log n \log \log n)$ and random Δ-regular graphs with $\Delta = \Omega(\log n)$.

Summary of Results

In §3.2, we analyze Structured Coupon Collection over n/b disjoint cliques, each of size b. In each trial, we choose $d \geq 1$ nodes independently and uniformly at random. If all the nodes in the corresponding cliques are covered, we do nothing. Otherwise, from among the chosen cliques containing an uncovered node, we select one at random and cover an uncovered node in it. We show that w.h.p., all the nodes are covered in $O(n)$ trials and

†By "with high probability" (w.h.p.), we mean "with probability at least $1 - O(n^{-\lambda})$ for an arbitrary constant $\lambda > 1$".

each of the first $\Omega(n)$ trials covers an uncovered node, for any choice of b and d satisfying $bd \geq c \log_2 n$ for a suitably large constant c.

In §3.3, we use the results proved in Section 3.2 to analyze a stochastic process for growing binary trees, thereby extending the suite of results known in this space. Adler *et al* [AHKV03] showed that if we repeatedly perform a random walk down the tree and split the shallowest of the "hypercubic neighbors" of the leaf node encountered, the resulting tree has leaves in $\Theta(1)$ levels. Abraham *et al* [AAA$^+$03] and Naor and Wieder [NW03] showed that the same property holds for trees resulting from the following process: at each step, we perform $c \log n$ random walks down the tree and split the shallowest leaf node encountered. Manku [M04] showed that if we perform one random walk, and split the shallowest leaf in the "vicinity" of the leaf node, where the vicinity has size $c \log n$, the resulting tree has leaves in at most three different levels, w.h.p. The process we analyze is a generalization of these two extremes (and includes both as special cases): we perform r random walks, inspect vicinities of size v for each of the r leaf nodes and then identify the shallowest leaf. We show that as long as $rv \geq c \log n$, the tree has leaves in at most four different levels, w.h.p. Analysis of the generalized process requires a new proof technique, for which we borrow ideas from [AHKV03] – the proof is different from the approaches taken by authors who analyze the extreme cases [AAA$^+$03, NW03, M04].

In §3.4, we show how balanced binary trees are useful for addressing the load balancing problem in Distributed Hash Tables (DHTs). The tradeoff between r, the number of random walks and v, the vicinity-size, is exploited to arrive at the optimal number of random walks required. We show that with $r = \sqrt{\log \log n}$ random walks, $O(\log n/\sqrt{\log \log n})$ messages are sufficient w.h.p., when a new member joins the DHT. Existing algorithms require $O(\log n)$ messages [AHKV03, M04] or $O(\log^2 n/\log \log n)$ messages [AAA$^+$03, NW03, KR04].

In §3.5, we review previous proposals for DHT load balancing.

In §3.6, we summarize our results and outline future directions for research.

3.2 Structured Coupon Collection over Cliques

Consider the problem of collecting b copies each of n/b coupons. In each trial, exactly one of the n/b coupons is chosen independently and uniformly at random. If b is a constant, the number of trials needed to collect all copies is *sharply concentrated* around $\frac{n}{b}(\ln \frac{n}{b} + (b-1) \ln \ln \frac{n}{b})$ [MR95]. In this Section, we study the following variant: we have to collect b

copies each of n/b coupons. In each trial, d coupons are chosen independently and uniformly at random but at most one of them can be retained to augment our collection: if we have already collected b copies of each of these d coupons, we do nothing; otherwise, from among the chosen coupons having less than b copies, we randomly select one to include in our collection. This process is equivalent to the process on cliques defined at the beginning of this Chapter. In this Section, our main results are that if $bd \geq c \log n$ for some suitably large constant c, then with high probability, (a) $O(n)$ trials suffice to collect b copies of all the coupons, and (b) each of the first $\Omega(n)$ trials increases the size of our collection.

3.2.1 Analysis

In terms of bins and balls, we have n/b bins, each with *capacity* b. In each trial, we choose d bins independently and uniformly at random. If all the d bins are full, we do nothing (the trial *fails*). Else, we select one of the non-full bins (from among the d choices) at random and place a ball into it. The first lemma below contains two useful forms of inequalities by Chernoff [C52] and Hoeffding [H63]. The second lemma helps us derive tail bounds for dependent binary random variables under certain conditions.

Lemma 3.2.1. *Let Z denote a random variable with a binomial distribution $Z \sim B(n,p)$.*

$$\text{For every } \lambda > 1, \qquad Pr[Z > \lambda np] < (e^{\lambda-1}\lambda^{-\lambda})^{np}.$$
$$\text{For every } a > 0, \qquad Pr[Z < np - a] < e^{-a^2/(2np)}.$$

Lemma 3.2.2. *Let $\omega_1, \omega_2, \ldots, \omega_n$ be a sequence of random variables. Let $Z_1, Z_2, \ldots, Z_n$ be a sequence of binary random variables, with the property that $Z_i = Z_i(\omega_1, \ldots, \omega_{i-1})$. Let $Z = \sum_{i=1}^{n} Z_i$. Then*

$$Pr[Z_i = 1 \,|\, \omega_1, \ldots, \omega_{i-1}] \leq p \quad \Rightarrow \quad Pr[Z \geq k] \;\leq\; Pr[B(n,p) \geq k].$$
$$Pr[Z_i = 1 \,|\, \omega_1, \ldots, \omega_{i-1}] \geq p \quad \Rightarrow \quad Pr[Z \leq k] \;\leq\; Pr[B(n,p) \leq k].$$

Theorem 3.1. *There exists a constant α such that, with high probability, all bins are full in αn trials, for any choice of b and d satisfying $bd \geq c \log_2 n$, where c is a sufficiently large constant.*

Proof. We will use the fact that $(1 - \frac{1}{x})^x < e^{-1} < (1 - \frac{1}{x})^{x-1}$ for all $x > 1$.

Let f denote the fraction of non-full bins at any time. Fraction f is non-increasing over time, and we divide the process into two phases: In Phase I, $f \geq 1/d$. In Phase II, $0 < f < 1/d$. The intuition underlying our analysis is as follows. In Phase I, many bins are non-full. Hence we make rapid progress in populating the bins, terminating the phase in $O(n)$ steps. In Phase II, progress is slow. However, from the perspective of an individual non-full bin, progress is fast enough to fill it in $O(n)$ steps.

Claim: Phase I terminates within $t_1 = (\frac{e}{e-1} + \epsilon_1)n$ trials, w.h.p., where ϵ_1 is a small constant.

Proof: At any time-step, the success probability, i.e., the probability that the ball gets placed into some non-full bin is at least $1-(1-f)^d > 1-1/e$. Let n_s denote the number of balls lying in various bins when Phase I terminates. Clearly $n_s \leq n$. Let T be the total number of trials in this phase and Y_t be the number of successes in the first t trials. $Y_t = \sum_{i=1}^{t} Z_i$ where Z_i is the indicator random variable corresponding to success in the i^{th} trial. Let ω_i denote the random choices available to the i^{th} ball. Then, $Pr[Z_i = 1|\omega_1,\ldots,\omega_{i-1}] \geq 1-1/e$. Using Lemma 3.2.2 , we can conclude that

$$Pr[T > \tfrac{n(1+\delta)}{1-1/e}] = Pr[Y_{\frac{n(1+\delta)}{1-1/e}} < n_s] \leq Pr[B(\tfrac{n(1+\delta)}{1-1/e}, \tfrac{e-1}{e}) < n_s] \leq Pr[B(\tfrac{n(1+\delta)}{1-1/e}, \tfrac{e-1}{e}) < n]$$

Using Lemma 3.2.1, the probability is less than $e^{-n\delta^2/2(1+\delta)}$, which is $o(1/n^2)$ when $\delta = \epsilon_1(1-1/e)$. Thus Phase I terminates within t_1 steps w.h.p.

Claim: Phase II terminates within $t_2 = (2e+\epsilon_2)n$ trials, w.h.p., where ϵ_2 is a small constant.

Proof: Let C denote a bin that is non-full at the end of Phase I. The probability that C is one of the d bins selected is $1-(1-b/n)^d > db/2n$. Given that one of the bins is C, the probability that each of the other $d-1$ bins is full is $(1-f)^{d-1} > 1/e$. Overall, the probability that C gets the ball in any time-step in Phase II is at least $db/2en$. As before, it follows from Lemma 3.2.2 that the number of balls in C stochastically dominates† the random variable $B(t_2, db/2en)$. Using $bd \geq c\log_2 n$, Lemma 3.2.1 yields that, in t_2 trials, C becomes full with probability $1-o(1/n^2)$. By taking the union bound over all the n bins, Phase II terminates within t_2 steps w.h.p.

Choosing $\alpha = (\frac{e}{e-1} + 2e + \epsilon_1 + \epsilon_2)$, we find that αn trials are sufficient to fill all bins w.h.p., where ϵ_1 can be made arbitrarily small, and ϵ_2 can be made small by choosing a large c. □

Note that Theorem 3.1 holds even if at any step, we choose a non-full bin (from among the d choices) *arbitrarily* (for example, in an adversarial fashion).

†A random variable X *stochastically dominates* random variable Y iff $Pr[X \geq r] \geq Pr[Y \geq r] \forall r \in \Re$.

Theorem 3.2. *With high probability, each of the first βn trials succeeds in placing a ball, for any $\beta < \frac{1}{2}$ and any choice of b and d satisfying $bd \geq c \log_2 n$, for a sufficiently large constant c.*

Proof. The proof follows from a series of four claims:

Claim: In any of the first βn trials, for any β, the probability that a specific bin receives a new ball is at most $\frac{b}{n(1-\beta)}$.

Proof: At any time-step, for a specific bin C,

$$\Pr[C \text{ is chosen}] = 1 - (1 - b/n)^d < db/n$$

Let f denote the fraction of non-full bins at any time-step. Then Pr[Ball is placed in $C \mid C$ is chosen] $= \sum_{i=1}^{d} (\frac{1}{i}) \binom{d-1}{i-1} f^{i-1} (1-f)^{d-i} = (\frac{1}{df}) \sum_{i=1}^{d} \binom{d}{i} f^i (1-f)^{d-i} = \frac{1-(1-f)^d}{df} < \frac{1}{df}$. At the end of the first βn trials, the fraction of full-bins is at most β. Therefore, at any earlier time-step, $f > 1 - \beta$. By conditioning on the number of non-full bins found in the d bins, we get

$$Pr[\text{Ball is placed in C} \mid \text{C is chosen}] < \frac{1}{d(1-\beta)}$$

Therefore, the probability that C receives a new ball is at most $\frac{db}{n} \cdot \frac{1}{d(1-\beta)} = \frac{b}{n(1-\beta)}$.

Claim: For any $\beta < \frac{1}{2}$, there exists a constant $\mu > 1$ such that the probability that a specific bin becomes full at the end of βn trials, is at most $1/\mu^\beta$.

Proof: From Lemma 3.2.2 and the previous claim, the random variable $B(\beta n, \frac{b}{n(1-\beta)})$ stochastically dominates the number of balls received by a specific bin C in βn trials. Using the Chernoff/Hoeffding inequalities in Lemma 3.2.1, the probability that C becomes full is at most $1/\mu^b$ where $\mu = (\frac{\beta}{1-\beta}) e^{(\frac{1-2\beta}{1-\beta})}$, and $\mu > 1$ iff $\beta < \frac{1}{2}$.

Claim: With high probability, the fraction of full bins at the end of βn trials is at most $1/\nu^b$, for some constant $\nu > 1$.

Proof: Let $\nu = \sqrt{\mu} > 1$. There are two cases:

a) $b < \log_\nu n - \log_\nu \log_\nu n$: Let I_i for $i = 1, \ldots, n/b$, denote a set of indicator variables, one per bin. The variable is 1 if the bin becomes full within βn trials. The set of variables are dependent but *negatively correlated* [DR98]. Therefore, for tail bounds on their sum, it suffices to replace them by a set of independent variables. The sum is dominated by the random variable $B(n/b, 1/\mu^b)$. Using the Chernoff/Hoeffding inequalities in Lemma 3.2.1, the number of full bins is at most $\frac{n}{b\nu^b}$ (where $\nu = \sqrt{\mu} > 1$) w.h.p., provided $b < \log_\nu n - \log_\nu \log_\nu n$.

b) $b \geq \log_\nu n - \log_\nu \log_\nu n$: Any process with $b \geq \log_\nu n - \log_\nu \log_\nu n$ dominates the corresponding process with $d = 1$. A simple application of Chernoff/Hoeffding inequalities in Lemma 3.2.1 shows that the first βn trials succeed w.h.p., for sufficiently large c.

Claim: Each of the first βn trials succeeds in placing a ball w.h.p., where $\beta < \frac{1}{2}$.

Proof: The fraction of full bins at the beginning of i^{th} trial, for any $i \leq \beta n$, is also at most $1/\nu^b$. Therefore, the i^{th} trial fails with probability at most $(1/\nu^b)^d = o(1/n^2)$, if c is sufficiently large. By taking the union bound over the first βn trials, we obtain that w.h.p., all of them succeed. □

The constant α in Theorem 3.1 can be improved (see Theorem 3.4 in Section 3.3). We suspect that further improvement is possible – a sharp threshold result should hold. Further, we speculate that Theorem 3.2 should hold for any $\beta < 1$, not just $\beta < \frac{1}{2}$. In Section 3.3, we use Theorems 3.1 and 3.2 to prove that binary trees resulting from a certain stochastic process are highly balanced.

3.2.2 Related Work

The classic balls-and-bins problem involves bins with infinite capacity and $d = 1$ (see Johnson and Kotz [JK77] or the book by Kolchin *et al* [KSC78]). Recently, there has been interest in the computer science community in analyzing the height of the fullest bin. With n bins and n balls, the height of the fullest bin is $\Theta(\log n / \log \log n)$ (see Gonnet [G81], Mitzenmacher [M96] and Raab and Steger [RS98]). For the case $d \geq 2$, a breakthrough was achieved by Azar *et al* [ABKU99] who showed that the height of the fullest bin is $\log \log n / \log d + \Theta(1)$, if the least-loaded bin among the d bins is chosen at each trial. For further results and a survey of proof techniques for $d \geq 2$, see Mitzenmacher *et al* [MRS01]. Our focus is on cliques, which are equivalent to bins with *finite* capacity. Moreover, we wish to bound the height of the bin with the *fewest* balls.

Structured Coupon Collection over graphs was defined by Adler *et al* [AHKV03] who proved that $O(n)$ steps suffice for covering all nodes of hypercubes, Δ-regular graphs with $\Delta = \Omega(\log n \log \log n)$ and random Δ-regular graphs with $\Delta = \Omega(\log n)$. Alon [A04] has shown that at least $n - \frac{n}{\Delta} + \frac{n}{\Delta} \log_e \frac{n}{\Delta}$ steps are necessary to cover all nodes for any Δ-regular graph. The processes analyzed in [A04, AHKV03] choose exactly one random node per trial. Our focus is on cliques but with multiple nodes chosen in each trial. In fact, we allow a tradeoff between the number of random choices and clique sizes by allowing any

$\langle b, d\rangle$ satisfying $bd \geq c \log n$ for an n-node graph. In Section 3.4, this tradeoff is exploited to derive the optimal number of random messages to be sent for load balancing in Distributed Hash Tables.

3.3 Balanced Binary Trees

In this Section, we study a variety of stochastic processes over binary trees which result in highly balanced trees. Balance is measured in terms of the number of different levels to which leaf nodes belong. We begin with some definitions:

Level and vicinity: In a binary tree, the *level* of a node is the length of the path from the root to that node. The root has level 0. There are at most 2^ℓ nodes at level ℓ. The *vicinity* of a node at level ℓ is the set of $v(\ell)$ nodes at level ℓ that have a common ancestor at level $\ell - \log_2 v(\ell)$.

Functions $[\![x]\!]$, r *and* v: Let $[\![x]\!] \equiv 2^k$ where integer k satisfies $2^{k-1} < x \leq 2^k$. Let $r : \mathbb{N} \to \mathbb{N}$ be a monotonically non-decreasing function, i.e., $r(\ell+1) \geq r(\ell)$. Let $v : \mathbb{N} \to \mathbb{N}$ be a function satisfying $v(0) = 1$ and $v(\ell) \leq 2^\ell$. Moreover, either $v(\ell+1) = v(\ell)$ or $v(\ell+1) = 2v(\ell)$. Thus $v(\ell)$ always equals some power of two.

In order to motivate the stochastic process that we analyze, we summarize related results established in earlier papers. Each paper studies a different stochastic process for growing the tree:

1. At each step, we carry out one random walk and split the leaf node encountered. Adler *et al* [AHKV03] and Naor and Wieder [NW03] show that leaf nodes belong to $\Theta(\log \log n)$ different levels w.h.p.
2. At each step, we carry out $c \log n$ random walks down the tree and split the shallowest leaf node encountered. Abraham *et al* [AAA$^+$03] showed that after n steps, the tree has leaves in $\Theta(1)$ different levels w.h.p. Naor and Wieder [NW03] analyze a similar stochastic process: we first estimate $\log n$ and then carry out $c \log n$ random walks.
3. Let $v(\ell) = [\![c\ell]\!]$, where c is a suitably-large constant. First, we carry out a random walk to reach a leaf node $\mathbf{r}$. If all nodes in the vicinity of $\mathbf{r}$'s parent are split, we split $\mathbf{r}$ itself. Else, we split one of the leaf nodes in the vicinity of $\mathbf{r}$'s parent. Manku [M04] showed that the leaf nodes in the resulting tree belong to at most three different levels w.h.p.
4. First, we carry out a random walk to reach a leaf node $\mathbf{r}$. We then identify the *shallowest*

hypercubic neighbor[†] of $\mathbf{r}$, splitting it into two. Adler *et al* [AHKV03] established that the resulting tree has leaf nodes in $\Theta(1)$ different levels.

3.3.1 A Stochastic Process for Growing Binary Trees

In this paper, we study a generalization of processes 2 and 3 above. We grow the binary tree in a randomized fashion by splitting some leaf node at each step, as follows:

> *We first carry out a random walk down the tree. Let ℓ denote the level of the leaf node encountered. We then carry out $r(\ell) - 1$ additional random walks, to obtain a set of leaf nodes X. For leaf node $\mathbf{x} \in X$, if all nodes in the vicinity of its parent are already split, we retain $\mathbf{x}$ in the set. Otherwise, we replace $\mathbf{x}$ by its parent. Let X' denote the new set thus obtained. Let ℓ' denote the level of the shallowest node in X'. We shrink X' to arrive at set $X'' \subseteq X'$, limited to nodes at level ℓ'. We then choose some $\mathbf{x}'' \in X''$ uniformly at random, and split an un-split node belonging to the vicinity of $\mathbf{x}''$.*

Different combinations of functions r and v result in different processes: Process 1 corresponds to $r(\ell) = v(\ell) = 1$. Process 2 is equivalent to $r(\ell) = c \log n$ and $v(\ell) = 1$. A variation of this process is $r(\ell) = c\ell$ and $v(\ell) = 1$. Process 3 amounts to $r(\ell) = 1$ and $v(\ell) = \llbracket c\ell \rrbracket$. Process 4 amounts to $r(\ell) = 1$ and $v(\ell) = \ell$ where the vicinity is defined to be the hypercubic neighbors of a node (see [AHKV03] for more details). Our interest in this paper lies in the following general condition: $\forall \ell : r(\ell)v(\ell) \geq c\ell$. This includes Processes 2 and 3 as special cases. Our main result is the following theorem:

Theorem 3.3. *For any combination of r and v satisfying $\forall \ell : r(\ell)v(\ell) \geq c\ell$, where c is a suitably large constant, the tree is highly balanced – leaf nodes belong to at most four different levels.*

The motivation for analyzing the generalized stochastic process are three-folds:

1. We are able to demonstrate that the binary tree is balanced for all combinations of $\langle r(\ell), v(\ell) \rangle$ ranging from $\langle 1, c\ell \rangle$ to $\langle c\ell, 1 \rangle$. In other words, randomness can be traded off for vicinity-size *smoothly.*

[†] Label the left and right branches of the tree with 0's and 1's respectively. Let the sequence of bits along the path from the root to $\mathbf{r}$ denote the ID of $\mathbf{r}$. A hypercubic neighbor of a leaf node is obtained by flipping a bit in the ID string, and identifying the leaf node with the longest matching prefix of the new string. Please see reference [AHKV03] for a pictorial definition and more details.

2. The generalized process requires a new proof technique, for which we borrow ideas from [AHKV03]. The proof involves the analysis of an interesting balls-and-bins problem as a sub-problem (Section 3.2).

3. In Section 3.4, we design a simple load-balancing scheme for Distributed Hash Tables in peer-to-peer systems. The scheme is based upon the generalized stochastic process. The smooth tradeoff between r and v allows us to identify the optimal number of messages necessary for load balance. We show that $O(\sqrt{\log\log n})$ random walks are sufficient, resulting in a total of $O(\log n/\sqrt{\log\log n})$ messages. The message complexity improves upon all previous schemes [AAA$^+$03, AHKV03, NW03, M04, KR04] for comparable load balance. See Section 3.4 for more details.

3.3.2 Proof of Theorem 3.3

Throughout this Section, we assume that $\forall \ell : r(\ell)v(\ell) \geq c\ell$ for a suitably large constant c.

Lemma 3.3.1. *Assume that the following three claims hold for some constants μ_1 and μ_2:*

1. When $n > \mu_1 2^L$, no leaf is at level L or less, w.h.p., where $2^a < \mu_1 < 2^{a+1}$ for some $a \geq 1$.

2. When $n < \mu_2 2^L$, no leaf is at level L or more, w.h.p., where $2^b < 1/\mu_2 < 2^{b+1}$ for some $b \geq 1$.

3. $\mu_1/\mu_2 < 2^{a+b+1}$.

Then with high probability, the leaves of the tree belong to most $a + b + 1$ different levels.

Proof. Let $2^k \leq n < 2^{k+1}$ for some integer k. There are three cases to consider:

a) $2^k \leq n \leq \mu_1 2^{k+1}$: Leaves belong to levels $[k - a, k + b]$ w.h.p.

b) $\mu_1 2^{k+1} < n < \mu_2^{-1} 2^{k+1}$: Leaves belong to levels $[k - a + 1, k + b]$ w.h.p.

c) $\mu_2^{-1} 2^{k+1} \leq n < 2^{k+1}$: Leaves belong to levels $[k - a + 1, k + b + 1]$ w.h.p.

In any case, the leaves are in at most $a + b + 1$ different levels. □

Consider the structured coupon collection process over a graph with $2^i/v(i)$ cliques, each of size $v(i)$. At each step, $r(i)$ random choices are made. Let $\mathcal{A}_i$ denote the process that terminates when all nodes in the graph have been covered. Let $\mathcal{B}_i$ denote the process that terminates when the first failure occurs, i.e., no new node could be covered. Let $\mathcal{A}^{(\ell)}$ and $\mathcal{B}^{(\ell)}$ denote series of processes $\langle \mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_\ell \rangle$ and $\langle \mathcal{B}_0, \mathcal{B}_1, \ldots, \mathcal{B}_\ell \rangle$, respectively.

In the remainder of the Section, we will use four constants: α, β, γ and δ. The first two are defined as follows: Let $\alpha 2^i$ denote an upper bound on the number of steps taken by $\mathcal{A}_i$

to terminate, with probability at least $1 - 1/poly(2^i)$ (see Theorem 3.1). Let $\beta 2^i$ denote a lower bound on the number of steps taken by $\mathcal{B}_i$ to terminate, with probability at least $1 - 1/poly(2^i)$. From Theorem 3.2, β can be set to any constant less than half. Constants γ and δ emerge in Lemmas 3.3.2 and 3.3.3 respectively. The interplay of all four constants will appear towards the end of this Section, when we prove Theorem 3.3.

Lemma 3.3.2. *$\mathcal{A}^{(k)}$ terminates in at most $\alpha(2+\gamma)2^k$ steps, w.h.p., where γ is an arbitrarily small constant.*

Proof. Let $j = \lceil \log_2 1/\gamma \rceil$, a constant depending upon γ. For process $\mathcal{A}_i$ where $0 \le i < k - \log_2 k - j$, we allocate $\alpha k 2^i$ steps. The probability that $\mathcal{A}_i$ does not terminate in $\alpha 2^i$ steps is at most $1/poly(2^i)$. Therefore, the probability that $\mathcal{A}_i$ does not terminates in $\alpha k 2^i$ steps is at most $(1/poly(2^i))^k = O(1/poly(2^k))$. The total number of steps we have allocated so far is $\sum_{i=0}^{i=k-\log_2 k-j-1} \alpha k 2^i < \alpha 2^{-j} 2^k \le \alpha\gamma 2^k$.

We now allocate $\alpha 2^i$ time-steps to each process $\mathcal{A}_i$ where $k - \log_2 k - j \le i \le k$. With probability at least $1 - 1/poly(2^i) = 1 - O(1/poly(2^k))$, process $\mathcal{A}_i$ terminates within $\alpha 2^i$ steps. The total number of steps is $\sum_{i=k-\log_2 k-j}^{i=k} \alpha 2^i < \sum_{i=0}^{i=k} \alpha 2^i < \alpha 2^{k+1}$.

The total number of steps is at most $\alpha(2+\gamma)2^k$. ☐

Lemma 3.3.3. *$\mathcal{B}^{(k)}$ takes at least $\beta(2-\delta)2^k$ steps to terminate, w.h.p., where δ is an arbitrarily small constant.*

Proof. Let $j = \lceil \log_2 1/\delta \rceil$, a constant depending upon δ. For $k-j-1 \le i \le k$, the probability that process $\mathcal{B}_i$ runs for less than $\beta 2^i$ steps is at most $O(1/poly(2^i)) = O(1/poly(2^k))$. Therefore, the series of processes $\langle \mathcal{B}_{k-j-1}, \mathcal{B}_{k-j}, \ldots, \mathcal{B}_k \rangle$ runs for at least $\sum_{i=k-j-1}^{i=k} \beta 2^i \ge \beta(2-\delta)2^k$ steps w.h.p. Thus $\mathcal{B}^{(k)}$ takes at least $\beta(2-\delta)2^k$ steps to terminate, w.h.p. ☐

Lemma 3.3.4. *When $n > \alpha(2+\gamma)2^L$, no leaf is at level L or less, with high probability, where γ is an arbitrarily small constant.*

Proof. We divide the growth of the tree into phases. Phase i is over (and phase $i+1$ starts) when no node at level i is a leaf node. Let T_i denote the time-step at which phase i terminates. To prove that no leaf is at level L or less, we will show that T_L is stochastically dominated by the time taken for $\mathcal{A}^{(L)}$ to terminate. The claim then follows from Lemma 3.3.2.

Let ℓ denote the level of the leaf node encountered in the first random walk down the tree. In phase i, all leaves are in level i or more. Therefore, $\ell \ge i$. Since function r is monotonically non-decreasing, $r(\ell) \geq r(i)$. Moreover, each vicinity that permits splitting of a leaf at level i, has size exactly $v(i)$, corresponding to a clique in process $\mathcal{A}_i$. Thus it follows that T_L is dominated by the time taken for $\mathcal{A}^{(L)}$ to terminate. □

Lemma 3.3.5. *When $n < \frac{1}{4}\beta(2-\delta)2^L$, no leaf is at level L or more, with high probability, where δ is an arbitrarily small constant.*

Proof. Consider the following variant of our algorithm: As soon as the *first* leaf at some level ℓ is created, we instantly create all leaf nodes at level $\ell - 1$ as well. This variant grows the tree faster than our original algorithm. Clearly, the variant is dominated by the original algorithm, in terms of the number of steps taken before creating the first leaf at level L. The variant is equivalent to process $\beta^{(L-2)}$, which runs for at least $\frac{1}{4}\beta(2-\delta)2^L$ steps (from Lemma 3.3.3). □

We are now ready to prove a claim slightly weaker than Theorem 3.3: we will establish that leaf nodes of the tree belong to at most five different levels. Let $\mu_1 = \alpha(2+\gamma)$ and $\mu_2 = \frac{1}{4}\beta(2-\delta)$. From Theorem 3.1, $\alpha = \frac{e}{e-1} + 2e + \epsilon_1 + \epsilon_2$, where ϵ_1 and ϵ_2 are arbitrarily small constants. It is possible to fix these constants so that μ_1 satisfies $2^2 < \mu_1 < 2^3$. Moreover, with a suitable choice of $\beta < 1/2$ (allowed by Theorem 3.2), and sufficiently small δ, we can arrive at a value for μ_2 that satisfies $2^2 < 1/\mu_2 < 2^3$, and $\mu_1/\mu_2 < 2^5$. From Lemma 3.3.1, it then follows that leaf nodes belong to at most five different levels.

To prove that leaf nodes belong to at most four different levels, as claimed in Theorem 3.3, we need a tighter version of Theorem 3.1, which we now prove.

Theorem 3.4. *With high probability, all bins are full in $\frac{9}{5}n$ trials, for any choice of d and b satisfying $db \geq c\log_2 n$ for a sufficiently large constant c.*

Proof. We treat $d \leq d_0$ (where d_0 is a constant to be defined later) as a special case. For a fixed value of b, the process with $d > 1$ choices dominates the process with a single choice ($d = 1$). A simple application of Chernoff/Hoeffding inequalities in Lemma 3.2.1 shows that if $\frac{9}{5}n$ balls were placed into $n/b \leq d_0 n/(c\log_2 n)$ bins (of unlimited capacity), each ball choosing a bin uniformly at random (i.e., $d = 1$), then every bin would get at least b balls w.h.p. for a suitably large value of c.

For the rest of the proof, we assume $d > d_0$. We divide the process into two phases as in the proof of Theorem 3.1. The analysis for Phase I is the same as before.

Claim: Phase II terminates within $t_2 = (\frac{1}{5} + \epsilon_2)n$ trials, w.h.p., where ϵ_2 is a small constant.

Proof: As before, for a specific bin C that is non-full at the end of Phase I, the number of balls in C stochastically dominates the random variable $B(t_2, db/2en)$. Choosing $d_0 = 28$ and using $db \geq c \log_2 n$, application of Chernoff/Hoeffding inequalities in Lemma 3.2.1 yields that, in t_2 trials, C becomes full with probability $1 - o(1/n^2)$. Taking the union bound over all the n bins yields the claim.

We need $(\frac{e}{e-1} + \frac{1}{5} + \epsilon_1 + \epsilon_2)n$ trials w.h.p., where ϵ_1 can be made arbitrarily small, and ϵ_2 can be made small by choosing a large c. The total is less than $\frac{9}{5}n$. $\square$

3.4 Load Balance in Distributed Hash Tables

In this Section, we bring out the connection between the stochastic process for growing binary trees (Section 3.3) and the load-balancing problem in Distributed Hash Tables (DHTs) in P2P networks. In order to make this Chapter self-contained, we briefly summarize DHTs.

3.4.1 Distributed Hash Tables: A Brief Summary

A DHT is maintained cooperatively, but in a decentralized fashion, by a large number of distributed hosts as follows. Imagine $[0, 1)$ as a circle with unit perimeter. Hosts are allowed to dynamically join the system (the set of participants is not fixed *a priori*). Upon joining, a host is assigned an ID in $[0, 1)$. At any instant, the current set of IDs *partitions* $[0, 1)$ into disjoint sub-intervals – each host is the *manager* of one such sub-interval. A host is connected with its successor and its predecessor along the circle with TCP connections, thereby forming a "ring" of hosts. The ring connections constitute the *short-distance* connections. Each host also makes *long-distance* TCP links with a few other hosts, as a function of its own ID. Taken together, the short- and long-distance TCP connections form the *overlay routing network* that is responsible for routing messages between hosts.

The problem of designing a good DHT routing network has received much attention recently. For our purposes, it suffices to treat the routing network as a black-box with the following property: Using the routing network, it is possible to send a message to the "manager" of a randomly-chosen point in $[0, 1)$ by paying a cost of R messages, with high probability. The earliest DHT routing networks were based on the hypercube and its variants.

These can route in $R = \Theta(\log n)$ messages, with only $\Theta(\log n)$ connections per node. Examples of these networks are Chord [SMK$^+$01, GM04], Pastry [RD01a] and Tapestry [ZHS$^+$04]. Later papers have shown that it is possible to achieve $R = \Theta(\log n/\log\log n)$ with the same number of connections. Examples of these networks are high-degree de Bruijn networks, as has been observed by several groups [AAA$^+$03, FG03, KK03, LKRG03, NW03], high-degree butterflies [KMXY03], Mariposa: a Kleinberg-style randomized butterfly [M03], and several other randomized networks that were analyzed in a recent paper [MNW04] (for example, randomized-Chord [ZGG03, GGG$^+$03], randomized-hypercubes [GGG$^+$03], Symphony [MBR03], skip-graphs [AS03] and SkipNet [HJS$^+$03]).

3.4.2 Balanced Binary Trees for Decentralized Load Balancing in DHTs

Upon arrival, a new host has to select an ID for itself[†]. DHTs are *decentralized* — there is no global knowledge of the current set of IDs. At any instant, any member of the ring can ascertain the IDs of k adjacent hosts in the ring by paying a cost of $2k$ messages, or it can identify the ID of a host that manages a random number in [0, 1), by paying a cost of R messages. A good ID selection algorithm should enjoy three properties: (a) simplicity and decentralization, (b) low-cost in terms of number of messages, and (c) small variation in partition sizes for load balance among managers. We will quantify variation in partition sizes by defining σ, the *partition balance ratio*, as the ratio between the largest and smallest partition sizes.

The relationship between binary trees (described in Section 3.3) and host IDs is as follows. Only leaf nodes of the tree correspond to host IDs. The internal nodes of the tree are conceptual. The sequence of 0s and 1s along the path from the root to a leaf node, treated as the binary expansion of a fraction in $[0, 1)$, constitutes the ID of that leaf.

A "random walk down the tree" is equivalent to identifying the manager of a point chosen uniformly at random from the interval $\mathcal{I} = [0, 1)$. We need R messages per random walk. Inspecting the "vicinity" of a leaf node or its parent (see Section 3.3 for a formal definition of vicinity) is equivalent to identifying whether the corresponding set of IDs along the circle exists or not. To make the inspection low-cost, we stipulate that each host maintain knowledge of its vicinity at all times. Thus when a new host is added to the ring (some leaf node splits in the corresponding tree), all other nodes in its vicinity are informed

[†]It is customarily assumed in DHT design that the new host "knows" one existing member of the ring at the outset.

of the arrival. By choosing $r(\ell) = \sqrt{\log \ell}$ and $v(\ell) = [\![c\ell/\sqrt{\log \ell}\,]\!]$, we obtain the following theorem:

Theorem 3.5. *A new host needs* $\Theta(\log n/\sqrt{\log\log n})$ *messages w.h.p. to obtain an ID, where* n *denotes the current number of hosts and* $R = \Theta(\log n/\log\log n)$*. The partition balance ratio is* $\sigma = \Theta(1)$ *w.h.p.*

Proof. From Theorem 3.3, the tree has leaves in at most four different levels w.h.p. With n leaf nodes, the level of any leaf is $\Theta(\log n)$ w.h.p. With $r(\ell) = \sqrt{\log \ell}$, the number of random walks down the tree is $\Theta(\sqrt{\log\log n})$. With $R = \Theta(\log n/\log\log n)$ messages per random walk, the total number of messages is $\Theta(\log n/\sqrt{\log\log n})$.

Whenever a new host is inserted, all other members of the vicinity it belongs to, are informed of its existence. Informing all members of a vicinity of size $v(\ell)$ requires at most $v(\ell)$ messages (by using only the short-distance "ring"-connections). With n leaf nodes, the level of a leaf node is $\Theta(\log n)$ w.h.p. With $v(\ell) = [\![c\ell/\sqrt{\log \ell}\,]\!]$, a vicinity has $\Theta(\log n/\sqrt{\log\log n})$ nodes, requiring as many messages.

Since leaf nodes are in $\Theta(1)$ different levels, $\sigma = \Theta(1)$ w.h.p. $\square$

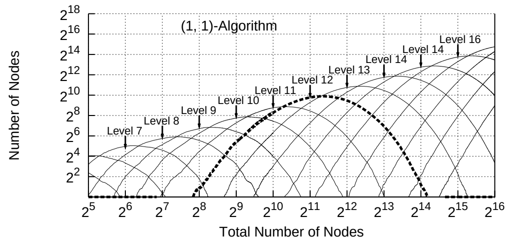
3.4.3 Lower Bound on σ with Balanced Binary Trees

How small a value of σ can possibly be realized in a decentralized setting? Ideally, we would like to have a distributed algorithm which ensures that the number of bits in any ID is either $\lfloor \log_2 n \rfloor$ or $\lceil \log_2 n \rceil$, when the current number of hosts is n. This goal seems unattainable for a decentralized algorithm because of the following intuition. When $n = 2^k - 1$, all leaf nodes in the corresponding tree should ideally be in levels $\{k-1, k\}$. However, with $n = 2^k + 1$, all leaf nodes should be in levels $\{k, k+1\}$. Therefore, a decentralized algorithm is likely to have leaves in at least three different levels, especially when n is close to a power of two[†].

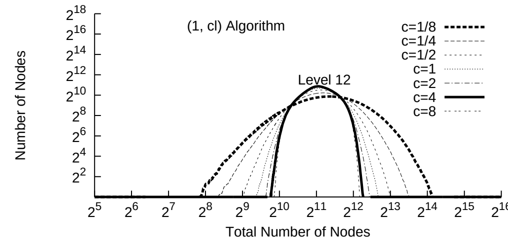
3.4.4 Experimental Results

Figure 3.1 studies various $\langle r, v\rangle$ schemes for growing binary trees. With $\langle r, v\rangle = \langle 1, 1\rangle$, host IDs belong to as many as six different levels when $n = 2^{11}$. With $\langle r, v\rangle = \langle 1, 4\ell\rangle$, host IDs are in only three different levels. Thus 8 appears to be a safely large value for constant c in the constraint: $\forall \ell : r(\ell)v(\ell) \geq c\ell$. Finally, five random walks are sufficient to obtain 3-level

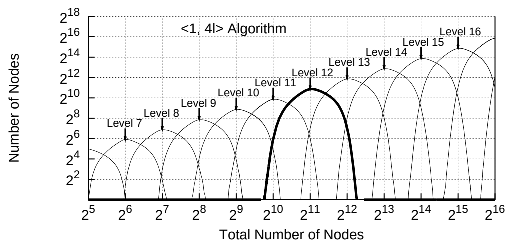
[†]A formal proof requires a precise *definition* of the notion of decentralization.



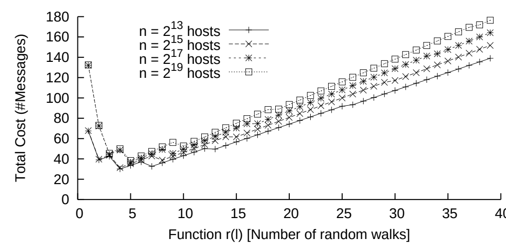
(a) ID distribution with $\langle 1, 1\rangle$ scheme.



(b) Effect of increasing c in $\langle 1, c\ell\rangle$ schemes.



(c) ID distribution with $\langle 1, 4\ell\rangle$ scheme.



(d) Total cost for varying # of random walks.

Figure 3.1: *In (a), the dotted curve shows the number of hosts with 12-bit IDs, as the total number of hosts increases over time, with the $\langle 1, 1\rangle$ scheme. The number of curves intersecting a vertical line equals the number of different levels at which leaf nodes exist, for that n. In (c), using $c = 4$, the $\langle 1, c\ell\rangle$ scheme results in 3-level trees. In (d), the four curves correspond to $n = 2^\ell$ hosts for $\ell = 13, 15, 17$ and 19 respectively. The total cost is $Rx + y$, where x is the number of random walks, $y = \llbracket 4\ell/x \rrbracket$, and $R = \ell/\log_2 \ell$ hops on average.*

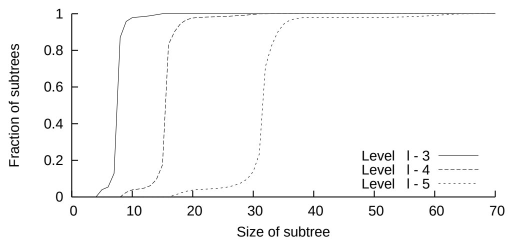
trees, when the number of hosts is $n = 2^{16}$. In terms of messages, this is superior to either of the two extremes: $\langle 1, c\ell\rangle$ and $\langle c\ell, 1\rangle$.

Figure 3.2(a) shows the distribution sub-tree sizes at levels $\ell - 3$, $\ell - 4$ and $\ell - 5$, with $n = 2^{16}$ leaf nodes. At level $\ell - 5$, over 99% of the sub-trees have 16 or more leaf nodes.

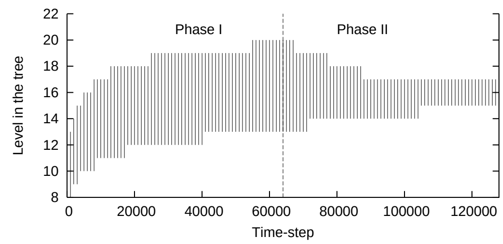
Figure 3.2(b) shows that our ID management algorithm restores balance quickly. If we carry out 64K insertions with the $\langle 1, 1\rangle$ scheme, followed by an equal numbers of insertions with the $\langle 4, \ell\rangle$ scheme, the resulting tree has leaf nodes in only three levels.

Virtual IDs were proposed in CFS [DKK$^+$01], where each host chose $\Theta(\log n)$ IDs for itself, each ID being a random number in $[0, 1)$. The scheme results in $\sigma = \Theta(1)$ at the cost of amplifying the number of overlay connections by $\Theta(\log n)$. In fact, σ can be made arbitrarily close to 1 by choosing a sufficiently large number of virtual IDs. Figure 3.3 shows that our algorithm requires far fewer virtual IDs to achieve a given value of σ.

Intuitively, we would expect that an increase in the number of virtual IDs per host would *always* diminish the variation in partition-sizes from the mean. However, Figure 3.3



(a) Cumulative distribution of sub-tree sizes ($n = 2^{16}$ hosts).



(b) Phase I: We use $\langle 1, 1\rangle$ scheme to carry out $64K$ insertions. Phase II: We use $\langle 4, \ell\rangle$ scheme for $64K$ interleaved insertions and deletions. Our algorithm restores balance quickly.

Figure 3.2: *Evaluation of our ID management algorithm.*

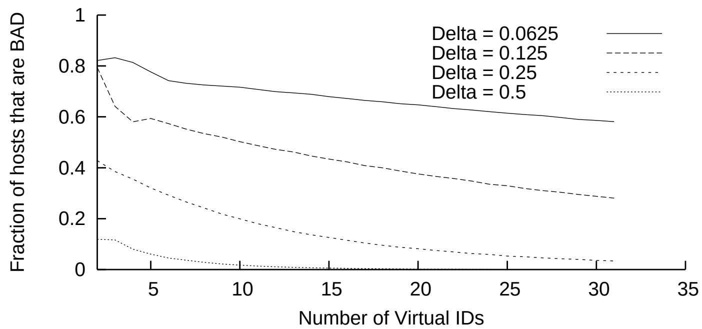
presents a curious phenomenon. Consider n, the number of hosts, when it is a power of two. If we make 2^i virtual IDs per host for some i, there are hardly any hosts whose sizes deviate from the mean. However, if we make $(1.5)2^i$ virtual IDs per host, the number of such hosts increases, since leaves are now (almost) evenly split between two successive levels.

We believe that the ability to tune $\sigma \leq 1 + \epsilon$, where ϵ is a user-defined parameter, *is* important. If we restrict ourselves to one ID per host, none of the algorithms based on binary-trees can achieve $\sigma < 2$. A proposal in reference [M04] is to create binary trees with the following twist: parents of leaves have degree between b and $2b - 1$, for some $b \geq 2$. By setting $b \approx \frac{1}{\epsilon}$, it is possible to achieve $\sigma = 1 + \epsilon$. However, at least $b - 1$ hosts require re-assignments per arrival/departure – a costly operation.

It would be interesting to devise ID management schemes to avoid the behavior in Figure 3.3 while using minimal re-assignments.

3.5 Related Work

Early DHT designs allowed each host to independently choose a number in $[0, 1)$ uniformly at random [SMK+01, ZHS+04, RD01a, RFHK01, FG03, KK03, MBR03, MNR02, HJS+03]. Such an algorithm is decentralized and requires zero messages to select an ID. However, $\sigma = \Omega(\log^2 n)$ [NW03] w.h.p., where σ denotes the *partition balance ratio*, defined as the ratio between the largest and the smallest partition sizes. King and Saia [KS04] recently established that $\sigma = \Theta(n \log n)$ w.h.p. If each host chooses a random number in $[0, 1)$



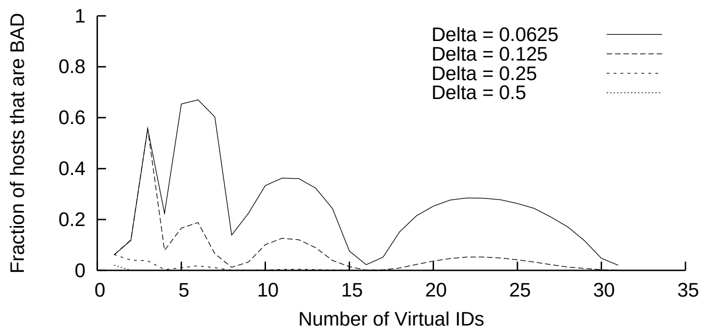


Figure 3.3: *A BAD host is one whose size lies outside the interval* $(1 \pm \delta)/N$*. The scheme for ID management is* $\langle 1, 1\rangle$ *for the top graph and* $\langle 4, \ell\rangle$ *for the bottom graph.*

and *splits* the partition the number falls into, σ diminishes to $\Theta(\log n)$ [AHKV03, NW03]. Further improvement is possible. If each host creates $\Omega(\log n)$ *virtual IDs* [DKK+01], σ reduces to $O(1)$. However, the number of overlay connections per host gets amplified by a factor of $\Omega(\log n)$ – this is costly because higher degree overlay networks require more resources for maintenance.

Two different approaches for ID management have recently been proposed, each of which guarantees $\sigma = \Theta(1)$ with only one ID per host. The first approach [NW03, AAA+03, KR04, M04] is overlay-independent while the second is overlay-dependent [AHKV03]. Naor and Wieder [NW03] and Abraham *et al* [AAA+03] proposed that a new host should choose $\Theta(\log n)$ random points from $[0, 1)$, identify the managers of these points and split the largest manager into two. Karger and Ruhl [KR04] proposed an elegant variation on the idea that supports departures as well, albeit at the cost of *re-assignment* of IDs of at most $O(\log \log n)$ hosts w.h.p. per arrival and departure. Adler *et al* [AHKV03] analyzed an overlay-dependent scheme that is specific to hypercubes. The idea is to identify the manager of a random point in $[0, 1)$, probe other managers it has established overlay connections with, and to split the largest of these managers into two. A scheme for handling departures exists, but it has not yielded to formal analysis yet. The idea in [AHKV03] had earlier been proposed as a heuristic in an early DHT paper [RFHK01]. Manku [M04] recently established that $\sigma \leq 4$ for the following scheme: a new host first randomly identifies the manager of a random number in $[0, 1)$, inspects $\Theta(\log n)$ managers in its "vicinity" and splits the largest manager. Departures are handled similarly and cause at most one host ID to be re-assigned. The message complexity for the schemes outlined above is either $\Theta(\log^2 n/\log\log n)$ messages [AAA$^+$03,NW03,KR04] or $\Theta(\log n)$ messages [AHKV03,M04], for networks with $R = \Theta(\log n/\log\log n)$.

Two additional approaches to load-balancing are as follows. Byers *et al* [BCM04] suggest a variant of the power-of-two choices paradigm (see Mitzenmacher *et al* [MRS01] for a survey). In their scheme, nodes continue to choose random numbers in $[0, 1)$ as their IDs. However, an object is stored at one out of several possible locations (determined by multiple hash-values of the object-name). A drawback of this idea is the overhead associated with multiple probes necessary when storing and retrieving objects. Godfrey *et al* [GLS$^+$04] take a systems approach, identifying the *run-time* loads on various nodes. They propose heuristics for re-distributing objects between pairs of lightly- and heavily-loaded nodes.

Load balance for range-queries over an ordered list of data elements has also received attention in the context of DHTs. The goal is to divide a sorted list of elements among a fixed number of blocks in the face of adversarial insertions and deletions of elements, such that the ratio of the largest and the smallest block is $\Theta(1)$. Essentially, two operations are allowed: (a) two adjacent blocks are re-balanced, or (b) two adjacent blocks are fused into one block while a large block is split into two. The cost associated with a fusion or a split is proportional to the number of data elements that have to *move*. Ganesan *et al* [GBGM04] present an efficient deterministic algorithm that guarantees a constant number of moves per data element, amortized over the lifetime of the data structure. If each block is assigned to one machine in a DHT, and if a skip-graph [AS03] is maintained over the starting keys of blocks, we obtain an efficient load-balanced range-queriable data structure. The scheme by Ganesan *et al* assumes that the largest (and the smallest) block can be efficiently discovered. It is not clear how this operation can be implemented in a *decentralized* fashion without causing congestion in a DHT. Karger and Ruhl [KR04] and Aspnes *et al* [AKK04] present efficient congestion-free algorithms that match large blocks with small blocks. Their algorithms are randomized, decentralized and handle a dynamic number of blocks.

3.6 Summary and Future Directions

We studied Structured Coupon Collection, a problem introduced by Adler *et al* [AHKV03], over n/b disjoint cliques with b nodes per clique. Nodes are initially uncovered. At each step, we choose d nodes independently and uniformly at random. If all the nodes in the corresponding cliques are covered, we do nothing. Otherwise, from among the chosen cliques with at least one uncovered node, we select one at random and cover an uncovered node within that clique. We showed that as long as $bd \geq c \log n$, $O(n)$ steps are sufficient to cover all nodes w.h.p. and each of the first $\Omega(n)$ steps succeed in covering a node w.h.p.. These results were then utilized to analyze a stochastic process for growing binary trees that are highly balanced – the leaves of the tree belong to at most four different levels w.h.p. The stochastic process constitutes the core idea underlying a practical P2P load balancing scheme that beats earlier proposals for the same, in terms of message complexity.

Possible directions for future work:

1. Our load-balancing algorithm does not address host departures. Only two known algorithms handle departures [KR04, M04]. Simulations show that a simple variation of the insertion algorithm maintains 3-level trees: *"A departed host is replaced by the deepest leaf in the union of vicinities probed."* It would be interesting to formally analyze this scheme.
2. If we could establish Theorem 3.2 for any $\beta \in (\frac{1}{2}, 1)$, we could show that leaf nodes belong to at most three different levels, which we believe is optimal. Note: The special case $d = 1$ corresponds to the algorithm in Section 2.3, for which Theorem 3.2 can easily be shown to hold for any $\beta \in (\frac{1}{2}, 1)$.
3. It would be interesting to analyze structured coupon collection over cliques when $bd \not\geq c \log n$, and to explore the impact of multiple choices ($d \geq 2$) over general graphs. Perhaps the results derived by Adler *et al* [AHKV03] can be extended to larger families of graphs with/without multiple choices per trial.

Chapter 4

Scalable and Dynamic Emulation of Network Families

In this Chapter, we describe the Emulation Engine of Dipsea, (see Figure 1.1 on page 2 for a block-diagram of its architecture). Dipsea consists of a dynamic set of managers, each manager having an ID in $\mathcal{I} = [0, 1)$. It is convenient to visualize $\mathcal{I}$ as a circle with unit perimeter. In Chapters 2 and 3, we described efficient decentralized ID Management algorithms that assign IDs to managers as they arrive and depart. As a function of their IDs, managers make links among themselves to form an overlay routing network. This network is used for routing messages and consists of two sets of links: (a) the *short-distance* links connect each manager with a few of its neighbors along the circle – these result in a *fault-tolerant ring*, and (b) the *long-distance* links connect each manager to a few non-neighbors along the circle – these result in an *efficient* routing network with *short routes*.

A common design goal is to make the long-distance links of the overlay routing network *mimic* or *look like* a well-known graph structure, *e.g.,* a hypercube, a butterfly network or a de Bruijn graph. These three basic graphs, along with several of their variants, have been well-studied by computer scientists since 1980's in the context of inter-connection networks for parallel machine architectures (see Leighton [L92] for theoretical foundations of this area, and Duato *et al* [DYN03] for practical design issues). Four challenges arise when we attempt to make the long-distance links mimic such graphs:

a) *Arbitrary Number of Nodes*: Hypercubes are defined for 2^k nodes, butterflies have $k2^k$ nodes, and de Bruijn graphs are defined for m^k nodes, where $k, m \geq 1$ are both integers.

However, in a DHT, the current number of nodes is not necessarily 2^k or $k2^k$ or m^k.

b) *Dynamism*: The set of nodes in a DHT changes over time as new managers arrive and existing managers depart. In contrast, the number of nodes in a parallel machine is fixed.

c) *Scale*: The number of nodes in a DHT exhibits large variation, spanning several orders of magnitude.

d) *Physical Network Proximity*: Since DHT nodes belong to different geographical regions of the world, the latency (or the ping-time) between a pair of randomly-chosen nodes is quite high. We would like to make almost all long-distance links low latency.

Recently, certain families of *random graphs* have been investigated for routing purposes (see Chapters 8 and 9 for more details). Some of these are defined for arbitrary integers whereas others are defined over successive powers of two. The same four challenges arise if we desire that the long-distance links mimic one of these random graphs. On the whole, the problem of mimicking a given family of graphs, deterministic or randomized, can succinctly be stated from the perspective of a manager:

DEFINITION (The Problem of Scalable and Dynamic Emulation of Network Families). *Assume that we would like to mimic a specific family of graphs, say the hypercube. Given a manager with a specific ID, which other managers should it make long-distance links with?*

The Emulation Engine addresses the above problem for two different distributions of IDs:

DEFINITION (RANDOM distribution of IDs). *Each manager chooses an ID independently and uniformly at random from $\mathcal{I}$.*

DEFINITION (BALANCED distribution of IDs). *Manager IDs correspond to leaf nodes of a binary tree whose leaf nodes belong to at most three different levels:* $[\log_2 n]$ *and* $[\log_2 n] \pm 1$*, where* $[x]$ *denotes the integer closest to* x*.*

RANDOM distribution of IDs results from the No Probes scheme (see Section 2.1), and is highly skewed. With n managers, the ratio between the largest and the smallest partition is $\Theta(n \log n)$ (see King and Saia [KS04]). BALANCED distribution of IDs results from ID Management algorithms in Chapters 2 and 3, where the IDs correspond to leaf nodes of a binary tree in which each internal node has exactly two children. If we label the left and right branches of internal nodes with 0 and 1 respectively, then the sequence of bits from the root to a leaf node, treated as a fraction in $\mathcal{I}$, constitutes the ID associated with that leaf. The resulting distribution of IDs is BALANCED.

Summary of Results

In §4.1, we show how families of graphs defined over successive powers of two can easily be emulated for BALANCED distribution of IDs. Each node has to make *three* sets of links.

In §4.2, we introduce the problem of *physical network proximity*, presenting a simple idea that works in conjunction with the emulation scheme in §4.1. In fact, we show that the issue of physical network proximity can be addressed *independent* of the family we wish to emulate.

In §4.3, we show how families of graphs defined over successive powers of two can be emulated with only *two* sets of links per node. For both RANDOM and BALANCED distribution of IDs, we require two abstractions: *network size estimation* and *clustering*.

In §4.4, we present related work.

In §4.5, we summary our results and outline directions for further research.

4.1 Emulation with Three Sets of Links

In this Section, we describe the Emulation Engine for BALANCED distribution of IDs. Let $\langle G_0, G_1, G_2, \ldots \rangle$ denote an infinite family of directed graphs where graph G_i is defined over 2^i nodes. Let $C(\mathbf{x})$ denote a *cluster* consisting of all managers whose IDs have prefix $\mathbf{x}$.

Consider a manager with an ℓ-bit ID $\mathbf{x}$. This node computes three B-values: $B_1 = \ell - 2$, $B_2 = \ell - 3$, $B_3 = \ell - 4$. It then establishes three sets of links: one set corresponding to $\mathbf{x}_1$, the B_1-bit prefix of its ID, another set corresponding to $\mathbf{x}_2$, the B_2-bit prefix of its ID, and finally, a set corresponding to $\mathbf{x}_3$, the B_3-bit prefix of its ID. Let $\mathbf{x}_1^1, \mathbf{x}_1^2, \ldots, \mathbf{x}_1^{i_1}$ denote the i_1 neighbors of label $\mathbf{x}_1$ in graph G_{B_1}. Let $\mathbf{x}_2^1, \mathbf{x}_2^2, \ldots, \mathbf{x}_2^{i_2}$ denote the i_2 neighbors of label $\mathbf{x}_2$ in graph G_{B_2}. Let $\mathbf{x}_3^1, \mathbf{x}_3^2, \ldots, \mathbf{x}_3^{i_3}$ denote the i_3 neighbors of label $\mathbf{x}_3$ in graph G_{B_3}. Then node $\mathbf{x}$ makes $i_1 + i_2 + i_3$ links with one member each of the following clusters: $C(\mathbf{x}_1^1), C(\mathbf{x}_1^2), \ldots, C(\mathbf{x}_1^{i_1})$, $C(\mathbf{x}_2^1), C(\mathbf{x}_2^2), \ldots, C(\mathbf{x}_2^{i_2})$, and $C(\mathbf{x}_3^1), C(\mathbf{x}_3^2), \ldots, C(\mathbf{x}_3^{i_3})$. For the other end of a link, any member of the destination cluster suffices. For example, a manager with ID 0.010111 would make links corresponding to $B_1 = 0.0101$, $B_2 = 0.010$ and $B_3 = 0.01$. When emulating hypercubes, these links would be established with any member of clusters with the following prefixes:

$$\begin{array}{cccc} 0.1101 & 0.0001 & 0.0111 & 0.0100 \end{array}$$
$$\begin{array}{ccc} 0.110 & 0.000 & 0.011 \end{array}$$
$$\begin{array}{cc} 0.11 & 0.00 \end{array}$$

Routing starts off along that set of links that correspond to the smallest B-value at the source. Routing switches to the next higher B-value if it encounters a node which believes in a different triple of B-values. In BALANCED distribution of IDs, IDs correspond to leaf nodes in a binary tree at levels $[\log_2 n]$ or $[\log_2 n] \pm 1$, where n is the total number of leaf nodes. Therefore, each cluster on $[\log_2 n] - 1$ or fewer bits is non-empty. Also, there are at most three different triples of B-values that various nodes believe in. Therefore, it is guaranteed that the values $[\log_2 n] - 1$ is a member of all the triples. Therefore, a message will eventually be delivered to a cluster on $[\log_2 n] - 1$ bits. At this point, only $\Theta(1)$ distance remains to the destination, which can be covered by using the short-distance links.
Notes:

1. Among the three sets of links, several links can be *shared* in the case of certain families of networks, for example, the hypercube and de Bruijn graphs. We illustrate the idea with an example. Consider the set of links established by a node with ID 0.010111 when it is emulating a hypercube. Since cluster $C(0.1101)$ is a subset of cluster $C(0.11)$, a link established with some member of $C(0.1101)$ suffices when it comes to link-establishment with some member of $C(0.11)$.

   For the butterfly and its variants, the three sets of links are indeed quite different.

2. Families of networks defined over non-powers of two are handled as follows. We transform the given family into another family defined over powers of two. Then the emulation technique developed so far is readily applicable.

4.2 Physical-Network Proximity

In Dipsea, participating hosts belong to different geographical regions of the world. Therefore, the latency (or the ping-time) between a pair of randomly-chosen nodes is quite high. If most of the long-distance links are high-latency, and if routes are $O(\log n)$ hops long (as in a hypercube, for example), then the total time taken to transmit any message to its destination would be quite large. Such high latencies would render the DHT practically unusable.

The problem of physical network proximity has been addressed for two specific routing networks: Chord and the hypercube. In both cases, the basic topology is altered by introducing randomization: Chord gets transformed into randomized-Chord [GGG+03,ZGG03], and the hypercube gets transformed into randomized-hypercube [GGG+03,CDHR03]. The key idea is to make the topology less rigid by introducing choices for every link that is established. This allows a manager to establish a link with the closest manager, from among the choices available. See Bamboo [RGRK04] and recent improvements to Chord [DLS+04] for performance numbers.

The methodology proposed for Chord and the hypercube raises some questions: How do we address the problem for other graph structures, *e.g.,* butterflies and de Bruijn graphs? How do we deal with graphs which are already randomized *e.g.,* Symphony [MBR03] or Viceroy [MNR02]? Do we have to deal with each graph structure on a case-by-case basis, or is there a generic technique that works for all graphs, deterministic and randomized?

In this Section, we present a generic scheme for incorporating physical network proximity into arbitrary graph families. The scheme is a simple extension of the idea presented in Section 4.1. We justify and validate our idea through a series of experiments, using actual traces of inter-host ping times.

4.2.1 The Power of 16 Choices

We begin by understanding the distribution of inter-host IP latencies on wide-area networks. In a recent measurement study on an Internet router-level topology, Zhang *et al* [ZGG03] observed that the latency expansion follows a power law, i.e., the number of hosts within latency x of any given host is roughly proportional to a polynomial in x. In fact, their measurements indicate that the power-law exponent is close to 1, suggesting that the world "looks like a ring" with respect to latency characteristics. Therefore, intuitively speaking, a host can find a "nearby" host by randomly sampling a very small number of hosts, and selecting the "closest" of these[†]. More precisely, let d be the exponent of the power-law expansion, and δ be the maximum latency between a pair of hosts. Then, if a host samples k other hosts independently and uniformly at random, the expected latency to the closest of these hosts is $O(\delta/k^{1/d})$ [ZGG03]. The improvement in latency drops off rapidly as k, the number of random samples, increases.

[†]This observation was first made by Zhang *et al* [ZGG03] who developed a random-sampling scheme to optimize Chord.

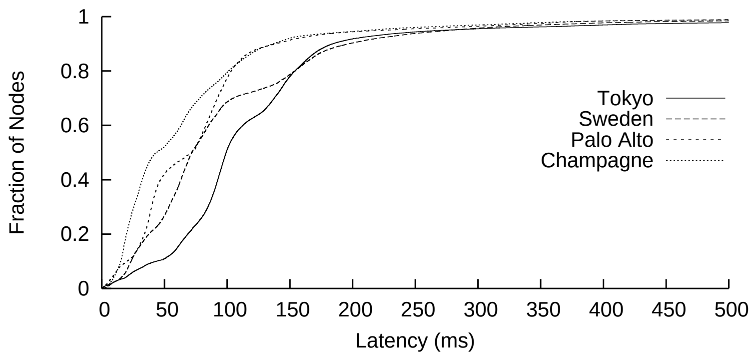


Figure 4.1: *CDF (Cumulative Distribution Function) of latencies as measured from four hosts in different parts of the world. We analyzed data obtained by the Skitter project [Ski].*

Analysis of Skitter Data: In order to understand the tradeoff between random sampling and reduction in inter-host latency, we analyzed data from the Skitter project [Ski]. The data consists of latency measurements from a fixed set of geographically dispersed hosts to a large numbers of end hosts, spanning most routable IP prefixes. We observe that all the CDF curves in Figure 4.1 initially exhibit a roughly linear behavior[†].

Random Sampling: Using Skitter data, we quantify the benefit of random sampling by plotting the expected latency to the closest of k randomly chosen hosts, in Figure 4.2 on the facing page. Observe that the latency drops very sharply as k increases beyond 1, showing that sampling even a small number of hosts is very beneficial. However, the benefits of sampling taper off rather quickly, irrespective of the location of the host performing the sampling. Even for the relatively "isolated" host in Japan, the expected latency drops from 146ms for one sample, to 39ms after eight samples, and to 24ms for sixteen samples. For the hosts in the United States, the latency drops even more sharply. We thus conclude the following:

> *(The Power of 16 Choices) In a global P2P system with geographically-diverse participants, a host is likely to find a "nearby" host by sampling 16 hosts at random. The utility of further sampling is marginal.*

We note that a sample of size 16 yields a nearby host *in expectation*. For high probability guarantees, we would have to oversample significantly. However, 16 samples is *good enough* for DHT routing, as we now show.

[†]We also observe non-linear step-like behavior which is not surprising, given that the world is clustered into continents separated by vast bodies of water.

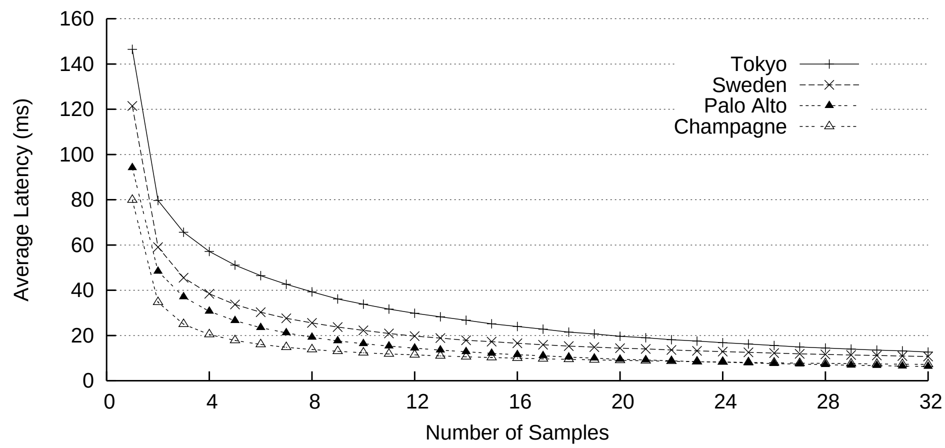


Figure 4.2: *Expected latency as a function of the number of hosts sampled, using data from the Skitter project [Ski]. The curve appears hyperbolic, as expected with exponent ≈ 1 in power-law expansion.*

4.2.2 Network-Proximity with BALANCED Distribution of IDs

Let us gain intuition into the relationship between long-distance links in Dipsea and physical-network proximity, by starting off with a simple idealized configuration: 2^k hosts, each host having a unique k-bit ID. Hosts lie in geographically disparate regions, and the sorted sequence of IDs is a random permutation of hosts. Let σ denote the average inter-host latency in the physical-network. Now if each host makes k overlay links corresponding to the hypercube, the average routing latency would be $\approx \frac{k}{2}\sigma$ because $\frac{k}{2}$ is the average path length in the hypercube and σ is the average IP-latency. The fundamental question is: How could we reduce average latency to 2σ or less? One possibility is to relax the definition of the hypercube [RD01a, ZHS$^+$04]: For $1 \leq i \leq k$, the i^{th} link would correspond to flipping the i^{th} most-significant bit along with *any* bit-combination in the low-order $k-i$ bits. The new network is *non-deterministic*. It introduces *choices* for each link – this gives a host the privilege to select a nearby host from among the choices available. The relaxed hypercube continues to guarantee worst-case routes of length $O(k)$. The Chord network can also be relaxed along the same lines [GGG$^+$03, ZGG03]. From among the choices available, a nearby host can be identified by *random sampling* at run time [ZGG03].

Limited Relaxation of the Hypercube: The relaxation of the hypercube, as described above, gives certain links *too many choices*. In Figure 4.2 , we saw that 16 samples is sufficient. This motivates the following experiment:

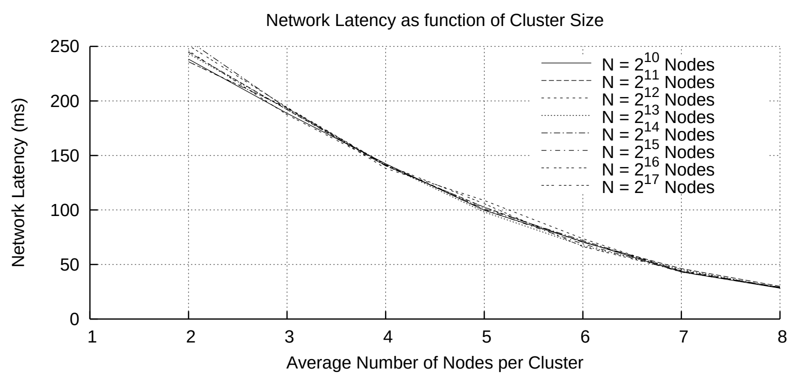


Figure 4.3: *Average latency as a function of cluster size.*

> We relax the hypercube such that for each $1 \leq i \leq k - c$, the i^{th} link corresponds to flipping the i^{th} bit and choosing any bit-combination in the c low-order bits, where c is a tunable parameter. It is easy to see that this network can be used to *correct* the top $k-c$ most-significant bits when routing. For the c low-order bits, we assume that the message is routed directly, thereby incurring a delay of σ on average. A relaxed version of CHORD can be defined along the same lines. Effectively, we have divided 2^k hosts into 2^{k-c} clusters, each cluster having 2^c hosts. Question: *How does average path latency vary as a function of cluster-size c?*

For the experiment, we need knowledge of pair-wise IP latencies for a large number of geographically distributed hosts. The Skitter project [Ski] provides latency measurements only from a small number of hosts and does not provide a matrix of pairwise latencies. We therefore resort to the standard GT-ITM topology generator [ZCB96], and assign latencies of 5ms, 20ms and 100ms to stub-stub, stub-transit and transit-transit links, in order to generate a latency model.

Figure 4.3 plots the equivalent of Figure 4.2 on the page before for the GT-ITM topology. The drop in average latency (resulting from choosing the closest among 2^c hosts) is not as spectacular as in Figure 4.2 on the preceding page. This is not surprising – it is well-known that the GT-ITM model results in exponential latency expansion and, thus, does not reflect real Internet latency distributions [GGG$^+$03, ZGG03], which have power-law expansion. However, we choose to utilize this model as a *conservative* evaluation of the utility of clusters. Using a model with power-law expansion would only serve to improve the results we report hereafter.

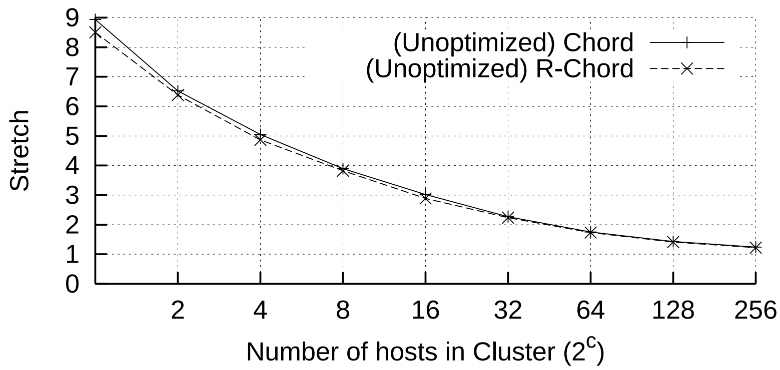


Figure 4.4: $N = 2^{16}$. *Increases in cluster-size result in diminishing returns.*

Figure 4.4 shows the result of the experiment on CHORD (and one of its variants called Randomized-Chord that we discuss in Chapter 8) It is clear that as c increases, the path latency is dominated by the last hop†. Figure 4.4 convinces us that our *limited relaxation of the hypercube is good enough* – with clusters of size 32, average route lengths are within 2σ. Figures 4.3 on the preceding page and 4.4 also illustrate that increases in cluster-sizes have diminishing returns. Figure 4.2 on page 75 reminds us that 32 is a conservative estimate. We believe that clusters of size 16 should be good enough in practice.

Optimizing the Last Hop: The average routing latency can be further reduced by taking advantage of replicas. Let us say that each host replicates its content at the $R-1$ hosts immediately succeeding it in circular ID-space. Then a host would maintain $R-1$ short-distance links with its successors. With an additional $\approx 2^c/R$ links, a host can reach a copy of every key-value pair within its cluster (we do not need 2^c links per host). For each of these additional links, there is a choice of R hosts to choose from. For example, with $c = 32$ and $R = 4$, each hosts needs eight additional links. With $R = 1$, the last hop costs σ. However, with $R = 2, 4, 6$, or 8, the cost of the last hop diminishes from σ down to 0.80σ, 0.64σ, 0.52σ, and 0.46σ, respectively.

4.2.3 Cluster-based Network-Proximity

The insight gained in the previous Section suggest a technique for introducing *flexibility* into *any* family of routing networks – deterministic or randomized – in a generic fashion. The

†The same phenomenon occurs in the relaxation of the hypercube we defined earlier due to limited choices available when flipping low-order bits [CDHR03, ZGG03, GGG$^+$03].

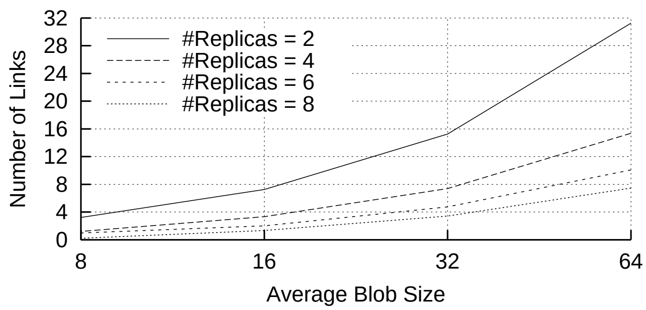


Figure 4.5: *Replication and Intra-Cluster Links*

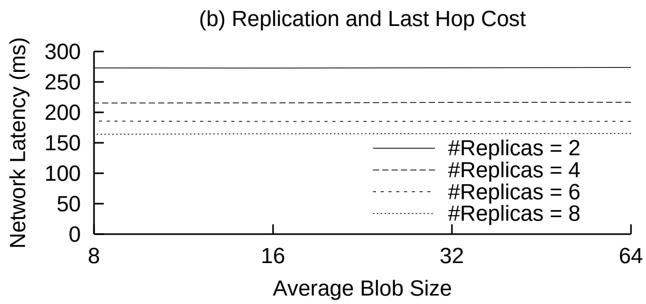


Figure 4.6: *Replication and last-hop cost*

design for a static collection of 2^k hosts can be summarized as follows: we divide the hosts into clusters of 16 hosts each. The top $k - c$ bits of a host's ID, where $c = 4$, constitute its cluster ID. On the basis of its cluster ID and the family of networks being emulated, a host figures out which other clusters it should have overlay connections with. These connections constitute the inter-cluster network. Once a message has reached some host belonging to the destination cluster, one last hop is required to reach the target host within the cluster. The last hop can be optimized, as discussed earlier.

As long as the sorted sequence of hosts in ID-space is a random permutation, a contiguous sequence of 16 hosts is *effectively* a random sample of size 16. As noted earlier (Figure 4.2 on page 75), a sample of this size yields a nearby host *in expectation*. For high probability guarantees, we would need many more samples. However, 16 samples is *good enough*, for three reasons: First, routes in DHTs consist of multiple hops, so it is the *sum of latencies over multiple-hops* that is of concern, not individual latencies. Second, the *average latency* over all host-pairs improves only marginally with additional samples. Third, on those rare occasions that all 16 samples are to geographically distant hosts, we rely on the redundancy in the family of routing networks to forward messages along alternative links.

4.2.4 Clusters in a Dynamic Network

In this Section, we adapt the basic idea in §4.2.3 to dynamic networks, highlighting issues that arise due to scale and dynamism. The crux of the matter lies in

(a) identifying which cluster(s) a host belongs to (this defines the set of out-going links), and

(b) for each out-going link, making sure that it points to a nearby host in the destination cluster.

Both these tasks are challenging. For example, agreement in cluster -sizes among large numbers of hosts is ruled out. Moreover, the number of clusters as well as their membership changes over time, possibly necessitating re-linking. We address these issues by exploiting certain properties of BALANCED distribution of IDs that are guaranteed by our ID management schemes in Chapters 2 and 3.

Cluster IDs: *How does a host figure out which cluster it belongs to?* A host with an ℓ-bit ID believes in three different k-values, where $k \in \{\ell-3, \ell-4, \ell-5\}$. It then constructs three sets of inter-cluster links, one corresponding to each value of k. For inter-cluster links, a host gets to choose from among 16, 32 and 64 different hosts, in expectation. Figure 3.2 on page 65(b) assures us that the deviation from these sub-tree sizes is small. Thus, with high probability, *there exists a complete inter-cluster network at some level $\hat{k}$ such that each cluster has at least* 16 *hosts*. Routing proceeds as described in §4.1. Indeed, for several networks like the hypercube, the three sets of links can be *shared*, as mentioned in §4.1.

Link Establishment: *How does a host identify a nearby host within a destination cluster?* One approach is to obtain the IP addresses of all (or 16 randomly chosen) hosts within a cluster, and ping each host individually to estimate the latency to that host. Currently, research is being carried out whether hosts can reliably be assigned "coordinates" or "vectors" in a metric space (see Vivaldi [DCD+04] or GNP [NZ02]) such that inter-host ping-times can be accurately deduced from these coordinates, obviating the need of actual pinging. With such coordinates in place, each host should learn the coordinates of 32 hosts in its vicinity. This enables fast identification of the nearest host in a cluster.

Re-linking Criteria: *How often does a host re-establish a link with some destination cluster?* Links are re-established only under two circumstances: (a) when a hosts fails/departs, any other host that had a link with the departed host, will attempt to find an alternative host within the same cluster to link to, and (b) when a host's ID changes in

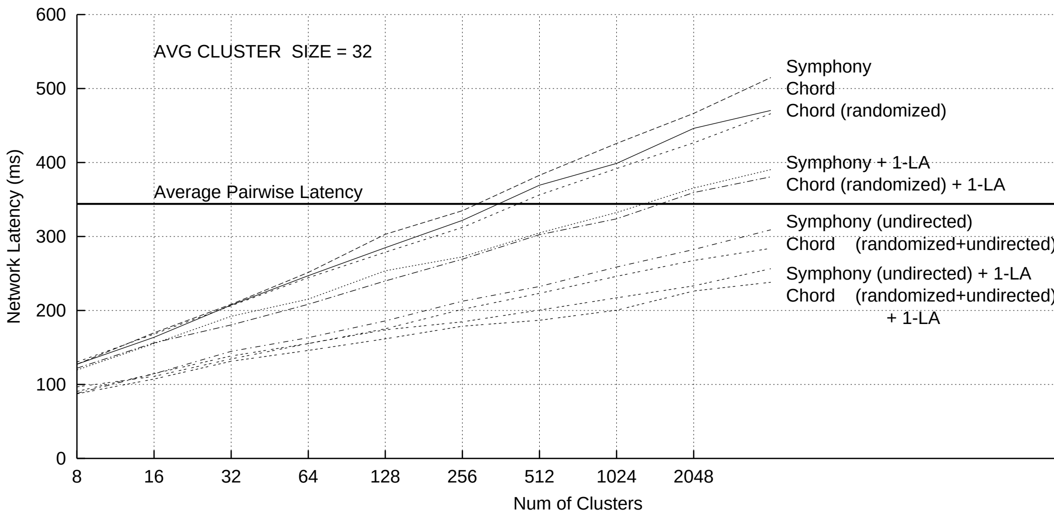


Figure 4.7: *Path latency for inter-cluster routing*

length, *all* three sets of links it possesses, are re-evaluated.

Note that the arrival of a host *does not* trigger a re-linking operation at any host that currently has links coming into the cluster that the new host just joined. The newly arrived host, along with one other host which was split (as a leaf node in the tree), create inter-cluster links. Having created its inter-cluster links (with destination clusters), a host remains oblivious of hosts that join any of the destination clusters.

The number of hosts affected by a host leaving in case (a) above depends upon the graph structure we are emulating. In several graphs, this number is $O(\log n)$. To understand when case (b) arises, observe that each host join or leave causes exactly one other host ID to change in length, in our ID Management scheme. Therefore, each host needs to re-establish its entire set of links only once per *half-life* of the system. As an optimization, this re-establishment of links may be done *lazily*, since it affects only the performance, not the correctness of routing.

Experiments: Figure 4.7 plots the stretch for a variety of networks, simulated on top of GT-ITM. Assuming 0 replicas and $64K$ hosts, we get a stretch of 1.6 with randomized-Chord using bi-directional links and 1-lookahead. With 4 replicas, the stretch reduces to 1.24 (because the last hop requires only 0.64σ latency on average).

4.2.5 Discussion

Our approach for incorporating network proximity awareness into long-distance links is unique because of its generality. Previous work has focused on specific topologies like Chord [GGG+03, ZGG03, DLS+04] or hypercubes [GGG+03, CDHR03, RGRK04]. In fact, we show that network proximity can be factored into the design *independent* of the choice of long-distance links, in a generic fashion. Moreover, our approach is quite simple to implement, as compared with other proposals that address network proximity [KR02, AMD04, HKMR04].

4.3 Emulation with only Two Sets of Links per Node

In this Section, we describe the design of an Emulation Engine for both RANDOM and BALANCED distribution of IDs. The improvement over the scheme in §4.1 is that each node has to make only *two* sets of links with other nodes instead of three.

The intuition underlying our scheme as follows: Emulation is challenging primarily due to two sources of uncertainty. First, the total number of participants is not known accurately to all managers. Second, the distribution of IDs is uneven. We address the first issue by developing a network-size estimation protocol. The second issue is addressed by clustering.

1. *Network Size Estimation*: Each manager runs a black-box whose goal is to maintain $\tilde{n}$, an estimate for n, the current number of managers. We stipulate that $\tilde{n} \in n/(1 \pm \delta)$ where δ is a tunable parameter. In practice, network size estimates could be derived from random sampling of partition sizes. Theoretically, a manager can derive an estimate by inspecting the distribution of IDs in its vicinity along the circle (see §4.3.2 and §4.3.3).

   A node with estimate $\tilde{n}$ computes two B-values: $B_1 = \lfloor \log_2(\tilde{n}/\alpha) \rfloor$ and $B_2 = \lceil \log_2(\tilde{n}/\alpha) \rceil$, where α depends upon the distribution of IDs. For RANDOM distribution, $\alpha = \frac{16}{\epsilon^2} \log \tilde{n}$. For BALANCED distribution, $\alpha = 4$. By fixing $\delta < 1/3$, we can ensure that there are no more than three different B-values that nodes believe in, and there exists some B-value that is common to all nodes.

2. *Clustering*: Let $\langle G_0, G_1, G_2, \ldots \rangle$ denote an infinite family of directed graphs where graph G_i has 2^i nodes. Let $C(\mathbf{x})$ denote a *cluster* consisting of all managers whose IDs have prefix $\mathbf{x}$.

Consider a node with label $\mathbf{x}$ which has computed two B-values: B_1 and B_2, as per the procedure outlined above. This node establishes two sets of links: one set corresponding to $\mathbf{x}_1$, the B_1-bit prefix of its ID and another set corresponding to $\mathbf{x}_2$, the B_2-bit prefix of its ID. Let $\mathbf{x}_1^1, \mathbf{x}_1^2, \ldots, \mathbf{x}_1^{i_1}$ denote the i_1 neighbors of label $\mathbf{x}_1$ in graph G_{B_1}. Let $\mathbf{x}_2^1, \mathbf{x}_2^2, \ldots, \mathbf{x}_2^{i_2}$ denote the i_2 neighbors of label $\mathbf{x}_2$ in graph G_{B_2}. Then node $\mathbf{x}$ makes $i_1 + i_2$ links with one member each of the following clusters: $C(\mathbf{x}_1^1), C(\mathbf{x}_1^2), \ldots, C(\mathbf{x}_1^{i_1})$ and $C(\mathbf{x}_2^1), C(\mathbf{x}_2^2), \ldots, C(\mathbf{x}_2^{i_2})$. For the other end of a link, any member of the destination cluster suffices.

Routing starts off along those links that correspond to the smaller of the two B-values at the source. Routing switches to the next higher B-value if it encounters a node which believes in a different pair of B-values. This B-value is guaranteed to be common to all nodes. Once the destination cluster has been reached, the distance remaining to the destination manager is $\Theta(1)$ hops for BALANCED distribution of IDs, and $\Theta(\log n)$ for RANDOM distribution of IDs. The remaining distance can be covered by following the short-distance links, or by setting up a small local routing network.

4.3.1 Flapping of Link-Sets

An issue that we have overlooked so far is the *flapping* of link-sets when $\tilde{n}$ hovers around a power of two. A simple modification prevents flapping. We stipulate that $\delta < 3 - 2\sqrt{2}$ for the Network Size Estimator. Consequently, the difference between $\log_2 \tilde{n}$ for two different managers is at most 1/2. We now introduce *hysteresis* to absorb small fluctuations in $\log_2 \tilde{n}$: A manager switches from a pair of B-values $\langle b, b+1\rangle$ to $\langle b+1, b+2\rangle$ only when $\log_2 \tilde{n}$ crosses the boundary $b + 1/2$ by *increasing* in value. The switch from $\langle b+1, b+2\rangle$ to $\langle b, b+1\rangle$ is made only when $\log_2 \tilde{n}$ crosses the boundary b by *decreasing* in value.

4.3.2 Network Size Estimation with RANDOM Distribution of IDs

In this Section, we develop a distributed scheme for estimating n, the current size of the network, for RANDOM distribution of IDs. Although different nodes arrive at different estimates of n, each estimate is guaranteed to lie in the range $n/(1 \pm \delta)$ with high probability[†] (w.h.p.) where $\delta \in (0, 1)$ is a user parameter. The intuition behind our scheme is captured by the following questions: Could a node with ID $\mathbf{x}$ deduce n by simply measuring the

[†]A guarantee is said to be with high probability if it fails with probability at most $1/n^c$ for some constant c.

density of IDs close to $\mathbf{x}$? How large a sub-interval suffices so that w.h.p., the actual number of IDs in the sub-interval does not deviate significantly from that expected?

Theorem 4.1 (Chernoff Inequality). *Let $X_1, X_2, \ldots, X_t$ denote independent Bernoulli variables with probability of success $p_i \in [0,1]$ for $1 \leq i \leq t$. Let $X = \sum_{i=1}^{t} X_i$ and $\mu = \mathbf{E}X = \sum_{i=1}^{t} p_i$. Then for any $0 < \epsilon < 2e - 1$, $Pr[X > (1+\epsilon)\mu] < \exp -\mu\epsilon^2/4$ and $Pr[X < (1-\epsilon)\mu] < \exp -\mu\epsilon^2/4$.*

Lemma 4.3.1. *Let n points be chosen independently, uniformly at random from the interval $[0,1)$. Let p_α be a random variable that equals the total number of points chosen in a fixed interval of size α. If $\alpha > (8\epsilon^{-2}\ln n)/n$, then $Pr[p_\alpha > (1+\epsilon)\mathbf{E}p_\alpha] < 1/n^2$ and $Pr[p_\alpha < (1-\epsilon)\mathbf{E}p_\alpha] < 1/n^2$.*

Lemma 4.3.1 follows immediately from Chernoff's Inequality. It suggests that we should measure the size of the interval spanned by $\Omega(\ln n)$ successive points and scale the observed density. Two issues remain: (a) How do we estimate $\ln n$ itself? (b) Exactly how many points suffice to arrive at an estimate for n lying in the range $n/(1 \pm \delta)$? Both issues are addressed by the following scheme: Consider a specific node with ID $\mathbf{x}$. Let n_i denote the number of nodes that share the top i (most significant) bits with $\mathbf{x}$. Node $\mathbf{x}$ identifies the largest ℓ such that $n_\ell \geq 16(1+\delta)\delta^{-2}\ln(2^\ell n_\ell)$.

Lemma 4.3.2. *(a) $\log_2(2^\ell n_\ell) > 0.5\log_2 n$ with probability at least $1 - 1/n^2$. (b) $2^\ell n_\ell$ lies in the interval $n/(1\pm\delta)$ with probability at least $1 - 2/n^2$.*

Proof. (a) If $2^k n_k \leq \sqrt{n}$, then $(1 - 1/2^k) \leq (1 - n_k/\sqrt{n})$. We assume that $n_k < n$ for otherwise, there is no error in estimate. Let us fix the IDs of the nodes that contribute to n_k. The probability that none of the remaining $n - n_k$ nodes chooses its id in the interval of size $1/2^k$ is given by $(1-1/2^k)^{n-n_k} \leq (1-n_k/\sqrt{n})^{n-n_k} \leq e^{-(n-1)/\sqrt{n}} < 1/n^2$ for large n.

(b) We know that $n_\ell \geq 16(1+\delta)\delta^{-2}\ln(2^\ell n_\ell)$. Part (a) above assures us that $\ln 2^\ell n_\ell > 0.5\ln n$ with probability at least $1-1/n^2$. Therefore, $n_\ell \geq 8\delta^{-2}(1+\delta))\ln n$ with probability at least $1-1/n^2$.

n_ℓ successive points are expected to lie in a sub-interval of size n_ℓ/n. However, we observed n_ℓ to lie in a sub-interval of size $1/2^\ell$. The probability that $1/2^\ell$ does not lie in the range $(1\pm\delta)n_\ell/n$ is given by $Pr[|1/2^k - n_\ell/n| > \delta n_\ell/n] \leq Pr[1/2^\ell < (1-\delta)n_\ell/n] + Pr[1/2^\ell > (1+\delta)n_\ell/n]$. We now prove that the first term is at most $1/n^2$. The proof for the second term is along similar lines.

Consider the probability $Pr[1/2^\ell < (1-\delta)n_\ell/n]$. This is identical to the probability $Pr[p_{(1-\delta)n_\ell/n} > n_\ell]$ (using the definition of p_α from Lemma 4.3.1). We can rewrite it as $Pr[p_{(1-\delta)n_\ell/n} > (1+\epsilon)(1-\delta)n_\ell/n]$ where $\epsilon = \delta/(1-\delta)$. From Lemma 4.3.1, this probability is less than $1/n^2$ as long as $\alpha = (1-\delta)n_\ell/n > (8\epsilon^2 \ln n)/n$, which is indeed true for $\epsilon = \delta/(1-\delta)$.

Consider the probability $Pr[1/2^\ell > (1+\delta)n_\ell/n]$. This is identical to the probability $Pr[p_{(1+\delta)n_\ell/n} < n_\ell]$ (using the definition of p_α from Lemma 4.3.1). We can rewrite it as $Pr[p_{(1+\delta)n_\ell/n} < (1-\epsilon)(1+\delta)n_\ell/n]$ where $\epsilon = \delta/(1+\delta)$. From Lemma 4.3.1, this probability is less than $1/n^2$ as long as $\alpha = (1+\delta)n_\ell/n > (8\epsilon^2 \ln n)/n$, which is indeed true for $\epsilon = \delta/(1+\delta)$. □

Theorem 4.2. *With probability at least $1-2/n$, the estimate of network size made by every node lies in the range $n/(1 \pm \delta)$.*

Proof. From Lemma 4.3.2 by summing the failure probability over all n nodes. □

The estimates of n made by all nodes lies in the range $n/(1 \pm \delta)$ w.h.p. On a log-scale in base two, the difference between the upper and lower bounds on the estimates is $\log_2[(1+\delta)/(1-\delta)]$. Setting $\delta < 1/3$ makes this difference less than 1. Let us label a node with estimate $\tilde{n}$ with a pair of B-values: $\langle \lfloor \log_2 \tilde{n} \rfloor, \lceil \log_2 \tilde{n} \rceil \rangle$. Then at most three different integers are used in labeling the nodes and at least one integer is common to all labels. Effectively, we have developed a decentralized scheme for reaching a rough consensus on the current size of the network, *i.e.,* the current number of managers along the circle.

Clustering

Lemma 4.3.3. *Let k be such that $2^k \leq (\epsilon^2 n)/(8 \ln n)$. With probability at least $1-2/n$, the number of points in each of 2^k equi-sized non-overlapping sub-intervals of $[0,1)$ lies in the range $(1 \pm \epsilon)n/2^k$.*

Proof. From Lemma 4.3.1, we conclude that with probability at least $1-2/n^2$, the number of points in a specific sub-interval lies in the range $(1 \pm \epsilon)n/2^k$. Summing the failure probability over $2^k \leq n$ intervals, we obtain the desired bound. □

Lemma 4.3.3 suggests a scheme for clustering. We label a node with estimate $\tilde{n}$ with a pair of integers $\langle B_1, B_2 \rangle$ where $B_1 = \left\lfloor \log_2(\epsilon^2 \tilde{n})/(16 \ln \tilde{n}) \right\rfloor$ and $B_2 = \left\lceil \log_2(\epsilon^2 \tilde{n})/(16 \ln \tilde{n}) \right\rceil$.

Assuming $\delta < 1/3$ in the estimation scheme, at most three B-values are used for labeling all nodes and at least one B-value is common to all labels. For each B-value used in a label, Lemma 4.3.3 assures us that each of 2^B clusters will be populated by $(1 \pm \epsilon)n/2^B$ node IDs w.h.p.

A family of networks over successive powers of two is emulated by constructing an *inter-cluster* and an *intra-cluster* network as follows. A node with label $\langle B_1, B_2 \rangle$ makes two sets of links. The first set corresponds to using B_1 most significant bits of its ID and assuming 2^{B_1} clusters. The second set corresponds to using B_2 most significant bits and assuming 2^{B_2} clusters. When establishing a particular link, a node can choose any node belonging to the destination cluster. Since at least one integer is common to all labels, there is at least one value of B such that the network over 2^k clusters is complete, *i.e.,* every node makes links with a B-bit ID.

Routing initially follows links corresponding to the smaller of the two B-values at the source. Along the way, routing switches to the next higher k-value if necessary. Upon reaching the destination cluster, intra-cluster routing is done by some local routing network. Each cluster has $\Theta(\log n)$ nodes. So if each node is connected to $\Omega(\log n)$ nodes along the circle (as advocated by Liben-Nowell *et al* [LNBK02] to address fault-tolerance concerns), intra-cluster routing takes $O(1)$ hops.

Remark: The paradigm of first routing to the destination cluster and then to a node within the cluster could also form the basis of a common proof technique for the analysis of existing routing networks like Chord, Koorde [KK03], D2B [FG03], which are defined for RANDOM distribution of IDs.

4.3.3 Network Size Estimation with BALANCED Distribution of IDs

For BALANCED distribution of IDs, a factor-4 approximation of n can be deduced from the number of bits in a manager's own ID. This follows from the fact that IDs correspond to leaf nodes of a binary tree in which all leaves are guaranteed to lie in levels $[\log_2 n]$ and $[\log_2 n] \pm 1$, w.h.p. Is it possible possible to improve upon this estimate? In general, how could a manager derive a sharp estimate $\tilde{n}$ satisfying $\tilde{n} \in n/(1 \pm \delta)$ for a fixed $\delta < 3 - 2\sqrt{2}$, from local measurements alone?

Theorem 4.3. *Imagine n manager arrivals where each manager chooses a random number in $\mathcal{I} = [0, 1)$. If $X(k)$ managers choose the same top k bits for their IDs such that $X(k) \geq 8(1 + \delta)\delta^{-2} \ln n$, then $2^k X(k) \in n/(1 \pm \delta)$ with probability at least $1 - 2/n^2$.*

Proof. Follows from Lemma 4.3.2(b). ☐

Theorem 4.4. *Each manager can estimate $\tilde{n} \in n/(1 \pm \delta)$ w.h.p., if c is large enough, by a modification of the ID selection algorithm in Section 2.3 (at no extra cost).*

Proof. We borrow terminology from our description of the scheme in §2.3. A new manager, upon joining the system, identifies the size of the sub-tree hanging below the *parent* of its active ancestor, and deduces $\tilde{n}$ from this size, as discussed below. The new manager then informs all other managers below the sub-tree about its estimate.

Let the newly-arrived manager correspond to a leaf node at level ℓ. Let $\mathbf{a}$ denote the active ancestor of the leaf node when it was created. Then $\mathbf{a}$ is at level $\phi(\ell)$. Consider $\mathbf{a}'$, the parent $\mathbf{a}$, which would be at level $\phi(\ell) - 1$. Let Z denote the total number of leaf nodes in the sub-tree rooted at $\mathbf{a}'$. Then $Z \geq \chi(\ell) = 2^{\lceil \log_2 \ell \rceil + c - 1} \geq \ell 2^{c-1}$. Now, $\ell \geq \log_2 n - 2$ since all leaves lie in the bottom three levels of the tree. Therefore, $Z \geq (\log_2 n - 2) 2^{c-1}$. Each of these leaves chose the same top $\phi(\ell) - 1$ bits for their ID. In the past, for each $i \in [0, \ell - 2]$, $\chi(i)$ managers chose their IDs such that the top $\phi(i)$ bits form a prefix of the ID of $\mathbf{a}'$. If we (conceptually) let each of these managers carry out further coin tosses so as to decide which one of them would eventually have gone on to share the top $\phi(\ell) - 1$ bits with $\mathbf{a}'$, we would augment Z to obtain a quantity Z'. We claim that $\tilde{n} = 2^{\phi(\ell)-1} Z'$ is an unbiased estimate for n. In fact, if we fix c to the smallest integer that satisfies $(\log_2 n - 2) 2^{c-1} \geq 8(1+\delta)\delta^{-2} \ln n$, we can apply Theorem 4.3 to claim that $\tilde{n} \in n/(1 \pm \delta)$ w.h.p. The construction in §4.3 works with any constant $\delta < 3 - 2\sqrt{2}$, which makes c a constant. ☐

4.4 Related Work

Network Size Estimation

Estimation of $\log n$ is a sub-problem that often emerges in the context of DHTs. Viceroy [MNR02], Symphony [MBR03] and Mariposa [M03] (see also Chapter 9) require the estimate for constructing the overlay routing network. CFS [DKK$^+$01] requires the estimate to establish $\Theta(\log n)$ virtual IDs per host. The ID Management algorithm by Naor and Wieder [NW03] needs to probe $\Theta(\log n)$ random points in $[0, 1)$. Assuming that each host independently chooses an ID from $[0, 1)$ uniformly at random, a *constant-factor* approximation of $\log n$ can be deduced from two successive IDs (see Viceroy [MNR02]). Recently, a new scheme for estimating $\log n$ within constant-factors has been devised by Horowitz and Malkhi [HM03b]. Our algorithm in §4.3.2 improves the estimate to *constant-additive* error in $\log n$.

Emulation of Network Families

Concomitant with our work, two other proposals for emulation of graph families have emerged: Naor and Wieder [NW03] propose the "Continuous-Discrete Approach" – the main limitation is that their approach does not handle butterfly networks, or randomized networks. Abraham *et al* [AAA$^+$03] have developed a scheme for emulating families of graphs. Members of a family are required to possess a certain kind of recursive structure that allows parent-child functions to have a property called "child-neighbor commutativity". The authors show that hypercubes, de Bruijn graphs and butterflies can be defined recursively so as to enjoy the property. Randomized networks have not been addressed.

Three big advantages of our emulation scheme in §4.1 are: (a) We can handle families of randomized routing networks, (b) The graph family is not required to possess any kind of recursive structure – in fact, the graphs over successive powers of two could be quite different, say a torus and a butterfly, and (c) Our scheme allows for physical-network proximity to be incorporated into the design *cleanly*.

4.5 Summary and Future Work

We described the design of an Emulation Engine for Dipsea. We showed how a variety of graph families could be emulated in a generic fashion. The Emulation Engine absorbs the complexity arising out of dynamism (arrivals/departure of hosts), scale (variation in the average number of hosts), and concerns of physical network proximity. We validated our design through a series of experiments using real-world latencies measured by the Skitter project [Ski] and by using the GT-ITM topology generator [ZCB96]. This leaves us free to explore the top-most module of Dipsea: Choice of Long-Distance Links (see Figure 1.1 on page 2). We devote Chapters 5 through 9 in understanding that module.

Future Work: It would be interesting to identify and to address practical issues that arise in an implementation of the Emulation Engine, on PlanetLab [CCR$^+$03], for example.

Chapter 5

Shortest Paths in Chord

In Chapters 5 through 9, we study the module "Choice of Long-Distance Links" in the Overlay Routing layer of Dipsea (see Figure 1.1 on page 2 for a block-diagram of its architecture). In this Chapter, we focus on Chord, a routing topology proposed in an early DHT design paper [SMK$^+$01].

5.1 Introduction

In the original paper [SMK$^+$01], Chord is defined over n nodes where each node chooses a random number in $[0, 1)$ as its ID. Edges in Chord are defined as follows: Imagine placing all nodes along a circle with unit perimeter. Then a node *manages* that portion of $[0, 1)$ that lies between its own ID and that of its clockwise successor along the circle. Each node creates edges with the managers of points which lie at the following distances in the clockwise direction: $\langle \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots \rangle$. Nodes in Chord represent hosts on the Internet and edges represent network connections. The edges in Chord should be treated as undirected because the corresponding TCP connections allow message delivery along both directions.

In this Chapter, we study an *idealized* version of Chord, which is defined over $n = 2^b$ nodes lying at the corners of a regular n-gon circumscribed by a unit circle. This is reasonable because the Emulation Engine for Dipsea (which we covered in detail in Chapter 4) allows emulation of arbitrary families of graphs defined over successive powers of two. With that in mind, we define Chord as follows:

DEFINITION (Chord). *Consider an undirected graph on 2^b nodes arranged in a circle. Nodes are labeled with b-bit identifiers from* 0 *through* $2^b - 1$ *going clockwise. An edge* (x, y) *exists iff* x *and* y *are* 2^k *positions apart on the circle for some* $k \geq 0$*, i.e.,* $|x - y|$ *equals either* 2^k *or* $2^b - 2^k$ *for some* $0 \leq k < b$.

Our goal in this Chapter is to characterize shortest paths in Chord by identifying routes of minimal length between pairs of nodes. From the perspective of shortest paths, the definition of Chord is deceptively simple; it hides a rich combinatorial structure, some of which we unearth in this Chapter.

In the standard Chord routing algorithm [SMK$^+$01], messages are forwarded along only those edges that diminish the clockwise distance by some power of two. Routing is clockwise and greedy, never overshooting the destination. If a message is destined for a node that is clockwise distance d away, routing is equivalent to performing left-to-right bit-fixing to convert the 1s in the binary representation of d to zero. For example, if d is 14 (1110 in binary), the standard Chord routing algorithm uses steps of 8, 4 and 2 in that order, thus converting the leftmost 1 in the remaining distance to a 0 at each step. The longest path has length b and the average path length is $b/2$. Clockwise greedy routing is non-optimal because it uses the non-diagonal edges in only one direction.

We show that shortest paths in Chord have a strong connection with the *Binary Subtraction Problem*: Given a positive integer d, find a pair of non-negative integers $\langle d', d'' \rangle$ such that the number of 1-bits in d' and d'' is minimal, subject to the constraint $d = d' - d''$. For example, the shortest route to cover clockwise distance 14 (1110 in binary) is to use a clockwise step of 16 in combination with an anti-clockwise step of length 2, which can be seen as an optimal way of expressing 14 as the difference of two numbers. Using the solution of the Binary Subtraction Problem as a building-block, we devise a procedure for identifying shortest paths between any pair of nodes in Chord. Analysis of this procedure shows that a shortest path in Chord has length at most $\lfloor b/2 \rfloor$ and that the average length of the shortest path is only $b/3 + \Theta(1)$.

Summary of Results

In §5.2, we study a simple but non-optimal variant of the standard Chord routing algorithm: each node chooses between the shorter of clockwise-greedy and anti-clockwise-greedy routes, on a per-destination basis. We show that the average path length drops to $b/2 - \sqrt{b/(2\pi)} + \Theta(1)$.

In §5.3, we expose the interplay between shortest path in Chord and binary subtraction.

In §5.4, we solve the Binary Subtraction Problem. In general, there is no unique solution. We present a non-deterministic procedure that generates all the optimal solutions.

In §5.5, we identify optimal routing algorithms for Chord. We show that Chord's diameter is $\lfloor b/2 \rfloor$. However, the average all-pairs shortest-path length is only $b/3 + \Theta(1)$. Interestingly, two simple algorithms that discover optimal routes can be encoded compactly by finite-state automata. The average shortest-path lengths are then computed by treating the automata as Markov Chains.

In §5.6, we extend our results to higher-base versions of Chord.

In §5.7, we describe the Hyperskewbe, a rather mysterious topology obtained by subtracting the hypercube edges from Chord but retaining the diameter edges.

In §5.8, we present related work.

In §5.9, we summarize our results and present possible directions for future research.

5.2 Choice of Two

A simple variant of the standard Chord routing algorithm is as follows: When node x wishes to send a message to y, it chooses the shorter of

(a) the clockwise greedy route to y, and

(b) the anti-clockwise greedy route to y's clockwise successor (namely $(y + 1) \bmod 2^b$), followed by an anti-clockwise step to y.

When both (a) and (b) have equal path lengths, we arbitrarily choose (b). The clockwise-greedy route in (a) considers only those edges that diminish the clockwise distance by some power of two. Similarly, the anti-clockwise-greedy route in (b) considers only those edges that diminish the anti-clockwise distance by some power of two. If we replace (b) by the more natural "anti-clockwise greedy route to y", Theorem 5.1 (see below) remains largely unchanged (the longest path is $\lfloor b/2 \rfloor$ instead of $\lceil b/2 \rceil$). However, analysis of link-congestion becomes involved (see reference [GM04] for details).

We now compute the average path length for Choice of Two. Let $d = (y - x + 2^b) \bmod 2^b$, the clockwise distance from x to y. Then the length of the clockwise-greedy route is $H(d)$ where $H(x)$ denotes the Hamming norm of x, i.e., the number of 1-bits in x. The length of the anti-clockwise greedy route, as described above, is $b - H(d) + 1$. The shorter of the two has length $\min\{H(d), b - H(d) + 1\}$. Let

$$S = \textstyle\sum_{d:H(d)\leq b} \min\{H(d), b - H(d)\} \qquad T = \sum_{i=0}^{b} i\binom{b}{i} = b2^{b-1}$$

The average shortest-path length for clockwise greedy routes is $T/2^b = b/2$. The average for Choice of Two is $S/2^b + \Theta(1)$. The quantity $S/2^b$ can be viewed in terms of a bins-and-balls problem: if b balls are thrown at random into one of two identical bins, then $S/2^b$ is the expected size of the smaller bin.

Lemma 5.2.1. *For* $0 \leq m < b$, $\quad \sum_{j=0}^{m}(b-2j)\binom{b}{j} = (m+1)\binom{b}{m+1}$

Proof. By induction. □

Theorem 5.1. *The average path length for the* Choice of two *algorithm is* $b/2 - \sqrt{b/2\pi} + \Theta(1)$ *while the longest path has length* $\lceil b/2 \rceil$.

Proof. The average path length is at most $S/2^b + 1$. If b is odd, $S = S_{odd}$, otherwise $S = S_{even}$, where

$$S_{odd} = \textstyle\sum_{i=0}^{(b-1)/2} i\binom{b}{i} + \sum_{i=(b+1)/2}^{b}(b-i)\binom{b}{i} \qquad S_{even} = \sum_{i=0}^{b/2} i\binom{b}{i} + \sum_{i=b/2+1}^{b}(b-i)\binom{b}{i}$$

Then

$$T - S_{odd} = \textstyle\sum_{i=(b+1)/2}^{b}(2i-b)\binom{b}{i} \qquad T - S_{even} = \sum_{i=(b+2)/2}^{b}(2i-b)\binom{b}{i}$$

Substituting $j = b - i$, we get

$$T - S_{odd} = \textstyle\sum_{j=0}^{(b-1)/2}(b-2j)\binom{b}{j} \qquad T - S_{even} = \sum_{j=0}^{(b-2)/2}(b-2j)\binom{b}{j}$$

Using Lemma 5.2.1, we get

$$T - S_{odd} = \tfrac{(b+1)}{2}\binom{b}{(b+1)/2} \qquad T - S_{even} = (b/2)\binom{b}{b/2}$$

We use Stirling's approximation, $\sqrt{2\pi x}(x/e)^x e^{1/(12x+1)} < x! < \sqrt{2\pi x}(x/e)^x e^{1/12x}$, to deduce

$$T - S_{odd} = \left[\sqrt{(b+1)/2\pi} + \Theta(1)\right] 2^b \qquad T - S_{even} = \left[\sqrt{b/2\pi} + \Theta(1)\right] 2^b$$

Thus, the average path length is $b/2 - \sqrt{b/2\pi} + \Theta(1)$. The longest path length is $\lceil b/2 \rceil$, corresponding to those values of d with $\lceil b/2 \rceil$ ones and $\lfloor b/2 \rfloor$ zeros. □

5.3 Optimal Routing and Binary Subtraction

Consider a route from node x to node y in Chord. Let us label each edge in the route by either a positive or a negative power of two as follows: Let c denote the clockwise distance traversed by following some edge. If $c = 2^{b-1}$, we label the edge arbitrarily as -2^{b-1} or $+2^{b-1}$. If $c = 2^k$ for some $0 \leq k < b-1$, then the label is $+2^k$. Otherwise, $c = 2^b - 2^k$ for some $0 \leq k < b-1$ and the label is -2^k. In other words, we mark clockwise and anti-clockwise usage of edges of length 2^k with positive and negative signs respectively.

Chord is symmetric with respect to every node. Therefore, a route corresponding to one of the shortest paths between x and y enjoys two properties: (a) no two labels along the path sum to zero, and (b) no label appears twice. Let d' be the sum of the positive labels, and d'' the sum of the absolute values of the negative labels. Recall that $H(x)$ is the Hamming norm of x, i.e., the number of ones in the binary representation of x.

Since each 1-bit in the binary representation of d' and d'' corresponds to some edge along the route, the length of the route is $H(d') + H(d'')$. Moreover, either $d = d' - d''$ or $2^b - d = d'' - d'$. To explain, when d' is greater than d'', their difference is the clockwise distance covered, and when d'' is greater than d', their difference is the anti-clockwise distance covered. We are now ready for a formal definition of the Chord routing problem.

Definition (Chord Routing Problem). *Given $b \geq 1$ and $0 < d < 2^b$, identify $\langle d', d'' \rangle$ such that $H(d') + H(d'')$ is minimal, subject to two constraints:*

(i) either $d = d' - d''$ or $2^b - d = d'' - d'$. *(ii) both $d', d'' \in [0, 2^b)$.*

In general, for fixed b and d, there are several optimal solutions for this problem. For example, if $b = 4$ and $d = 0110$, then $\langle 0110, 0000 \rangle$, $\langle 1000, 0010 \rangle$ and $\langle 0000, 1010 \rangle$ are three different optimal solutions. Moreover, a specific $\langle d', d'' \rangle$ actually encodes a *set* of optimal routes in Chord because any permutation of edges corresponding to 1s in $\langle d', d'' \rangle$ covers clockwise distance d.

It turns out that the Chord Routing Problem can be reduced to the following problem:

Definition (Binary Subtraction Problem). *Given positive integer d, find a pair of non-negative integers $\langle d', d'' \rangle$ such that $H(d') + H(d'')$ is minimized, subject to the constraint $d = d' - d''$.*

Both Binary Subtraction and Chord Routing produce a tuple $\langle d', d'' \rangle$ as output. However, the two problems differ in two respects, both involving the number of bits b. First, Binary Subtraction has only one constraint $d = d' - d''$ whereas Chord Routing has a choice: either $d = d' - d''$ or $2^b - d = d'' - d'$. Second, in Chord Routing, both d' and d'' are restricted to lie in the range $[0, 2^b)$. There is no such restriction in Binary Subtraction.

Procedure OPT_ROUTE(b, d) below shows how the Chord Routing Problem can be solved by using procedure OPT_SUBTRACT(d), which solves the Binary Subtraction Problem.

```
PROCEDURE OPT_ROUTE(b, d)
    IF (d < 2^{b-1})
        ⟨d', d''⟩ ← OPT_SUBTRACT(d)
        OUTPUT ⟨d', d''⟩
    ELSE
        ⟨d', d''⟩ ← OPT_SUBTRACT(2^b − d)
        OUTPUT ⟨d'', d'⟩
```

The next two Lemmas establish the correctness of procedure OPT_ROUTE. We will discuss procedure OPT_SUBTRACT in detail in §5.4.

Lemma 5.3.1. *If* $\langle d', d''\rangle =$ OPT_SUBTRACT(d) *and* $1 \le d \le 2^{b-1}$*, then* $\langle d', d''\rangle$ *is a valid solution for the Chord Routing Problem with parameters* b *and* d*.*

Proof. The proof is in two parts: First, we show that when $1 \le d \le 2^{b-1}$, then constraint *(i)* in the definition of the Chord Routing Problem can be simplified by dropping the latter condition. Second, we show that constraint *(ii)* can be dropped altogether. The resulting problem is simply the Binary Subtraction Problem with input d.

(a) Let $\mathcal{S}(b, x)$ denote the set of all tuples $\langle x', x''\rangle$ such that $x = x' - x''$ and both $x', x'' \in [0, 2^b)$. We will now show that if $d \le 2^{b-1}$, then for any $\langle d', d''\rangle \in \mathcal{S}(b, 2^b - d)$, $\left\langle 2^{b-1} + d'', d' - 2^{b-1}\right\rangle \in \mathcal{S}(b, d)$. We will also that $H(d') + H(d'') = H(2^{b-1} + d'') + H(d' - 2^{b-1})$ Together, the two claims will prove that for $1 \le d \le 2^{b-1}$, we can simplify constraint *(i)* in the Chord Routing Problem by dropping the second condition.

Consider $\langle d', d''\rangle \in \mathcal{S}(b, 2^b - d)$. By definition, $2^b - d = d' - d''$. Re-arranging the terms, we get $d = (2^{b-1} + d'') - (d' - 2^{b-1})$. By assumption, $d \le 2^{b-1}$. Therefore, $2^b - d \ge 2^{b-1}$, in which case, $d' \ge 2^{b-1}$ and $d'' < 2^{b-1}$. In other words, the "diagonal" edge is necessarily a part of d' and definitely not in d''. Now, consider the tuple $\left\langle 2^{b-1} + d'', d' - 2^{b-1}\right\rangle$. Loosely speaking, this tuple can be obtained from $\langle d', d''\rangle$ by "flipping" the direction of the diagonal. Since

$d' \geq 2^{b-1}$ and $d'' < 2^{b-1}$, it follows that both members of the tuple $\left\langle 2^{b-1} + d'', d' - 2^{b-1} \right\rangle$ lie in the interval $[0, 2^b)$. Thus $\left\langle 2^{b-1} + d'', d' - 2^{b-1} \right\rangle \in \mathcal{S}(b, d)$. It is easy to see that $H(d') + H(d'') = H(d' - 2^{b-1}) + H(2^{b-1} + d'')$.

(b) Since $d \leq 2^{b-1}$, if $\langle d', d'' \rangle$ is a solution of OPT_SUBTRACT(d), then both $d', d'' \in [0, 2^b)$. Therefore, constraint *(ii)* in the Chord Routing Problem can be dropped. □

Lemma 5.3.2. *If $\langle d', d'' \rangle$ is a solution for the Chord Routing Problem with parameters b and d, then $\langle d'', d' \rangle$ is a solution for the Chord Routing Problem with parameters b and $2^b - d$.*

Proof. Follows from the problem definition. □

Next, we present the solution to the Binary Subtraction Problem in §5.4. In §5.5, we will derive two simple, intuitive algorithms to directly discover optimal routes in Chord. In fact, these algorithms are simple enough to be described as finite-state automata.

5.4 Solving the Binary Subtraction Problem

Figure 5.1 on the next page shows pseudo-code for procedure OPT_SUBTRACT(d), which solves the Binary Subtraction Problem. The output tuple $\langle d', d'' \rangle$ is produced bit-by-bit, right to left. In fact, the procedure is non-deterministic and can produce *all* optimal pairs for a given value of input d.

OPT_SUBTRACT *scans* the binary representation of d, right to left, producing the corresponding bits of $\langle d', d'' \rangle$ along the way. As long as the current bit is 0, the output is $\langle 0, 0 \rangle$ since both bits at that position ought to be 0 in $\langle d', d'' \rangle$ for optimality. The current bit becomes 1 when $d \bmod 2 = 1$. Then the output must be either $\langle 1, 0 \rangle$ or $\langle 0, 1 \rangle$. Depending upon whether we choose $\langle 1, 0 \rangle$ or $\langle 0, 1 \rangle$, the algorithm continues scanning the remainder, i.e., $(d-1)/2$ or $(d+1)/2$ respectively. The crux of the matter lies in deciding which of the two possible outputs to produce when $d \bmod 2 = 1$. OPT_SUBTRACT provides a precise characterization of how the decision should be made. The decision involves regular expressions (see the book by Hopcroft, Motwani and Ullman [HMU00], for example).

SUFFIX($x, \mathcal{R}$) denotes a predicate that takes a positive integer x and regular expression $\mathcal{R}$ as input. Let $\mathbf{x}$ denote the binary representation of x in the form of a string with a leading 1. Then SUFFIX($x, \mathcal{R}$) evaluates to true iff string $00\mathbf{x}$ satisfies regular expression $(0+1)^*\mathcal{R}$.

```
PROCEDURE Opt_Subtract (d)
WHILE  (d > 0) {

    WHILE (d mod 2 = 0) {
        d ← d/2
        OUTPUT ⟨0,0⟩
    }

    t := { ⟨1,0⟩            if Suffix(d, 0(01)*01)
         { ⟨0,1⟩            if Suffix(d, 1(10)*11)
         { ⟨1,0⟩ or ⟨0,1⟩   otherwise

    d := { (d − 1)/2   if t = ⟨1,0⟩
         { (d + 1)/2   if t = ⟨0,1⟩

    OUTPUT  t
}
```

Figure 5.1: OPT_SUBTRACT(d) *solves the Binary Subtraction Problem. It outputs* <u>*all*</u> $\langle d', d'' \rangle$ *where* $d = d' - d''$ *and* $H(d') + H(d'')$ *is minimal.*

In other words, SUFFIX checks whether some suffix of $00\mathbf{x}$ satisfies $\mathcal{R}$. Prepending a pair of zeros to $\mathbf{x}$ might seem unnatural. However, it simplifies proofs by reducing the number of cases that we have to consider.

We now prove that this algorithm produces optimal solutions for the Binary Subtraction Problem. A reader interested in the algorithms for optimal routing in Chord, rather than in proofs of their optimality, may wish to proceed directly to §5.5.

Lemma 5.4.1. *If* $x \bmod 2 = 1$ *then exactly one of the following four predicates is true:*

(a) SUFFIX$(x, 0(01)^*01)$	*(c)* SUFFIX$(x, 00(10)^*11)$
(b) SUFFIX$(x, 11(01)^*01)$	*(d)* SUFFIX$(x, 1(10)^*11)$

Proof. SUFFIX$(x, \mathcal{R})$ is true iff string $00x$ satisfies $\mathcal{R}$. If $x \bmod 2 = 1$, then exactly one of the following two mutually exclusive conditions holds:

$00x$ *ends in* 01: Either (a) or (b) is true, depending upon whether a pair of zeros or a pair of ones is encountered when scanning $00x$ from right to left.

$00x$ *ends in* 11: Either (c) or (d) is true, depending upon whether a pair of zeros or a pair of ones is encountered first when scanning $00x$ from right to left, and ignoring the right-most bit. ∎

The non-determinism of OPT_SUBTRACT stems from the fact that we assign $t := \langle 0,1\rangle$ or $t := \langle 1,0\rangle$ arbitrarily if the suffix of d is neither 0 nor $0(01)^*01$ nor $1(10)^*11$. In other words, whenever d satisfies conditions (b) or (c) in the above lemma, the algorithm has a choice in producing its output. For example, any of the following five outputs can be produced for input $d = 1110011010$:

d'	10000011010	10000100010	10000100000	10000000010	10000000000
$-d''$	- 00010000000	- 00010001000	- 00010000110	- 00001101000	- 00001100110
d	1110011010	1110011010	1110011010	1110011010	1110011010

Proof of Optimality

We begin by defining functions Φ and G over the set of positive integers.

Φ: For $d \geq 1$, $\Phi(d) = \min[H(d')+H(d'')]$ over all possible $d', d'' \geq 0$ such that $d = d'-d''$.

G: We first define G for strings that satisfy the regular expression $1(1+0)^*$. Later, we will extend the definition to include positive integers. For string $\mathbf{x}$ that satisfies regular expression $(1^+0)^+1$, let g denote the number of "groups" of 1s in x; if all groups are singleton ones, $G(\mathbf{x}) = g$, otherwise, $G(\mathbf{x}) = g+1$. For example, $G(110111) = 2+1 = 3$, $G(101) = 2$ and $G(10111) = 2+1 = 3$. For any $\mathbf{x}$ that satisfies $(0+1)^*1(0+1)^*$, we claim that it is possible to write $\mathbf{x} = (0^*)\sigma_1(000^*)\sigma_2(000^*)\ldots\sigma_\ell(0^*)$ uniquely, where each sub-string $\sigma_1, \sigma_2, \ldots, \sigma_\ell$ satisfies the regular expression $(1^+0)^+1$. We are essentially breaking up the string $\mathbf{x}$ whenever we encounter consecutive zeros, and we are ignoring leading and trailing strings of zeros. Now, $G(\mathbf{x}) = \sum_{i=1}^{\ell} G(\sigma_i)$. For positive integer x, $G(x) = G(\mathbf{x})$ where $\mathbf{x}$ denotes the string representing x in binary.

Theorem 5.2. *For $d \geq 1$, $\Phi(d) = G(d)$. If $\langle d', d''\rangle$ is produced by* OPT_SUBTRACT(d) *as output, then $d = d' - d''$ and $H(d') + H(d'') = G(d)$.*

Proof. Proof by induction on integer d. *Base case:* $d = 1$. *Induction step:* Consider $d > 1$. Assuming that the claim holds for all integers less than d, there are two cases:

- $d \bmod 2 = 0$

  Clearly, $\Phi(d) = \Phi(d/2)$ and $G(d/2) = G(d)$. By induction hypothesis, $\Phi(d/2) = G(d/2)$. Thus $\Phi(d) = G(d)$. OPT_SUBTRACT indeed outputs $\langle 0,0\rangle$ when $d \bmod 2 = 0$.

- $d \bmod 2 = 1$

  If $d = d' - d''$, then exactly one of d' and d'' has 1 in the last bit. Therefore, for $d > 1$,

$$\Phi(d) = 1 + \min[\Phi((d-1)/2), \Phi((d+1)/2)]$$

  By induction hypothesis,

$$\Phi((d-1)/2) = G((d-1)/2)$$
$$\Phi((d+1)/2) = G((d+1)/2)$$

  Therefore,

$$\Phi(d) = 1 + \min[G((d-1)/2), G((d+1)/2)] \tag{5.1}$$

  The following identities are easily seen to hold:

$$\text{a) } \textsc{Suffix}(x, 0(01)^*01) \Rightarrow \begin{cases} G((x-1)/2) = G(x) - 1 \\ G((x+1)/2) = G(x) \end{cases}$$

$$\text{b) } \textsc{Suffix}(x, 11(01)^*01) \Rightarrow \begin{cases} G((x-1)/2) = G(x) - 1 \\ G((x+1)/2) = G(x) - 1 \end{cases}$$

$$\text{c) } \textsc{Suffix}(x, 1(10)^*11) \Rightarrow \begin{cases} G((x-1)/2) = G(x) \\ G((x+1)/2) = G(x) - 1 \end{cases}$$

$$\text{d) } \textsc{Suffix}(x, 00(10)^*11) \Rightarrow \begin{cases} G((x-1)/2) = G(x) - 1 \\ G((x+1)/2) = G(x) - 1 \end{cases}$$

  From Lemma 5.4.1, exactly one of the four SUFFIX predicates holds for d. Thus

$$\min[G((d+1)/2), G(d-1)/2)] = G(d) - 1 \tag{5.2}$$

  Eq (5.1) and Eq (5.2) together yield $\Phi(d) = G(d)$.

The correctness of OPT_SUBTRACT is clear if we rewrite the assignment to t as follows:

$$t := \begin{cases} \langle 1, 0 \rangle & \text{if } G((d-1)/2) < G((d+1)/2) \\ \langle 0, 1 \rangle & \text{if } G((d-1)/2) > G((d+1)/2) \\ \langle 1, 0 \rangle \textit{ or } \langle 0, 1 \rangle & \text{if } G((d-1)/2) = G((d+1)/2) \end{cases}$$

In fact, it can be shown that OPT_SUBTRACT(d) produces *all* possible tuples $\langle d', d'' \rangle$ such that $d = d' - d''$ and $H(d') + H(d'') = G(d)$. □

5.5 Optimal Routing Algorithms for Chord

We present two deterministic algorithms for solving the Chord Routing Problem that we defined in §5.3. For fixed values of b and d, exactly one output tuple $\langle d', d'' \rangle$ is produced. Both algorithms run the automaton in Figure 5.2 for exactly b steps.

5.5.1 Right-to-Left Chaining

The binary representation of d, possibly padded with leading 0s to make it exactly b bits long, is fed as input right-to-left. The output tuple $\langle d', d'' \rangle$ is also generated right-to-left. The start state is S_0. Each transition produces a pair of bits as output, the first bit for d' and the second for d''. All thin edges produce $\langle 0, 0 \rangle$ as output. Thick edges ($S_0 \xrightarrow{1} S_1$ and $S_2 \xrightarrow{0} S_3$) require a *lookahead*: If the next input bit is 0, the output is $\langle 1, 0 \rangle$. If the next input bit is 1, the output is $\langle 0, 1 \rangle$. If there is no next input bit, i.e., the input string just terminated, the output is arbitrarily chosen as $\langle 0, 1 \rangle$ or $\langle 1, 0 \rangle$. Observe that the traversal of a thick edge corresponds to exactly one 1-bit in either d' or d''.

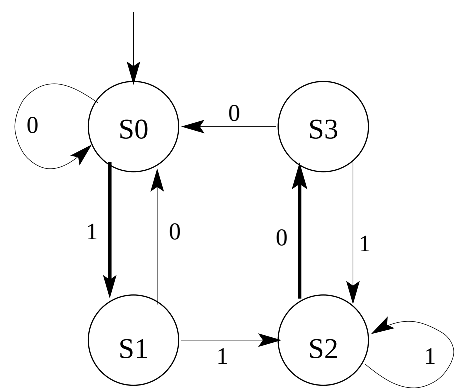


Figure 5.2: *A state machine for solving the Chord Routing Problem.*

Further intuition into Right-to-Left Chaining is gained by understanding the algorithm in terms of two ideas: *chain fixing* and *chain coupling*.

Chain fixing is simple: a distance corresponding to a chain of at least two 1s can be covered using a combination of only two steps: a short backward step and a long forward step. For example, a distance of 7 (000111 in binary) in a 64-node graph can be covered with a forward step of 8 combined with a backward step of 1. Chain fixing applies only to chains of length at least two. A chain consisting of a solitary 1 is handled by taking a single forward step.

Chain coupling comes into play when two chains are separated by a solitary zero. We explain the idea with an example. Let us say we had to cover clockwise distance 238(011101110). We know that the last five bits can be fixed by a backward step of 2 and a forward step of 16. However, we observe that taking the backward step of 2 leads to a distance 240(011110000), in which the bit corresponding to the intended forward step of 16 becomes part of a longer chain of 1s. Thus, instead of a forward step of 16, we can instead use a *backward* step of 16 followed by a forward step of 256 to fix this entire chain of 1s.

The behavior of the finite-state automaton in Figure 5.2 on the page before can be understood in terms of chain-fixing and chain-coupling. The automaton starts in State S_0, moving between S_0 and S_1 to fix singleton ones. Transition $S_1 \xrightarrow{1} S_2$ marks the onset of chain-fixing because two consecutive ones were just encountered in the input. Transition $S_2 \xrightarrow{0} S_3$, followed by $S_3 \xrightarrow{1} S_2$ marks the onset of chain-coupling. Transition $S_3 \xrightarrow{0} S_1$ signals the end of chain-coupling since two consecutive zeroes were just encountered, taking us to the start state S_0. If the input string happens to terminate in state S_1 or S_3, the output can be chosen arbitrarily as $\langle 1, 0\rangle$ or $\langle 0, 1\rangle$. The choice is immaterial because 1s in the most significant bit-positions of d' and d'' correspond to clockwise distances 2^{b-1} and $2^b - 2^{b-1}$ respectively, which are equivalent.

Examples:

d	d'	d''
0001	0001	0000
0011	0100	0001
0111	0000	1001

d	d'	d''
101010	101010	000000
110101	000101	010000
000111	001000	000001

Observe that the automaton solves $\textsc{Opt\_Subtract}(d)$ when $d < 2^{b-1}$. The deterministic algorithm is equivalent to replacing the two lines in Figure 5.1 on page 96 that assign values to t and d, as follows:

$$t := \begin{cases} \langle 1, 0\rangle & \text{if } \textsc{Suffix}(d, 01) \\ \langle 0, 1\rangle & \text{if } \textsc{Suffix}(d, 11) \end{cases}$$

$$d := \begin{cases} (d-1)/2 & \text{if } t = \langle 1, 0\rangle \\ (d+1)/2 & \text{if } t = \langle 0, 1\rangle \end{cases}$$

5.5.2 Left-to-Right Bi-directional Greedy

Thanks to Qixiang Sun, a colleague at Stanford University, for suggesting this algorithm. The idea is simple: Node x chooses the edge that takes the message the closest, in terms of absolute distance on the circle, to y. This algorithm can also be encoded using the automaton in Figure 5.2 on the page before. The input string $d[1..b]$ is treated as a bit-array of length b and is processed from `left` to `right`. The start state is S_0. The output is stored in bit-arrays $d'[1..b]$ and $d''[1..b]$, both of which are initialized to all-zeros. The output arrays get altered only when a thick edge is traversed. Let us assume that the thick edge was traversed due to the i^{th} input bit, where $1 \leq i \leq b$.

$S_0 \xrightarrow{1} S_1$: `if` $(i = b)$ or $(d[i+1] = 0)$
`then set` $d'[i] \leftarrow 1$
`else set` $d'[i-1] \leftarrow 1$

$S_2 \xrightarrow{0} S_3$: `if` $(i = b)$ or $(d[i+1] = 0)$
`then set` $d''[i-1] \leftarrow 1$
`else set` $d''[i] \leftarrow 1$

5.5.3 Proof of Optimality

Lemma 5.5.1. *Let* $\mathbf{d}$ *denote the binary representation of integer* $d \geq 1$. *The automaton traverses exactly* $G(d)$ *thick edges if string* $0\mathbf{d}$ *is fed right-to-left.*

Proof. Let $0\mathbf{d} = 0\sigma_1(00^*0)\sigma_2(00^*0)\sigma_3 \ldots \sigma_\ell(0^*)$ where each σ_i satisfies the regular expression $(1^+0)^+1$. We claim that the number of thick edges traversed when processing $0\mathbf{d}$ is $G(d) = \sum_{i=1}^{\ell} G(\sigma_i)$. For convenience, we rewrite $0\mathbf{d} = (0\sigma_1 0)0^*(0\sigma_2 0)0^*(0\sigma_3 0)0^* \ldots (0\sigma_\ell)0^*$. The automaton traverses no thick edges while processing the sub-strings corresponding to 0^*. For each $0\sigma_i$, the automaton performs one state transition from S_0 to S_1 for each of the initial singleton 1s, one state transition from S_0 to S_1 for the first of a non-singleton group of 1s, and one state transition from S_2 to S_3 at the end of every group of 1s thereafter. Observe that if the input string does not have a leading zero, i.e., the last input bit fed to the automaton is a 1, the automaton could potentially terminate in state S_2 without traversing a thick edge for the last group of 1s. □

Lemma 5.5.2. *For* $1 \leq d < 2^b$,

$$\begin{aligned} d \leq \left\lfloor 2^b/3 \right\rfloor \quad &\Rightarrow \quad G(d) = G(2^b - d) - 1 \\ \left\lfloor 2^b/3 \right\rfloor < d \leq \left\lfloor 2^{b+1}/3 \right\rfloor \quad &\Rightarrow \quad G(d) = G(2^b - d) \\ d > \left\lfloor 2^{b+1}/3 \right\rfloor \quad &\Rightarrow \quad G(d) = G(2^b - d) + 1 \end{aligned}$$

Proof. For $d = 2^{b-1}$, the Lemma is true. For $d \geq 1$ and $d \neq 2^{b-1}$, the b-bit strings representing d and $2^b - d$ satisfy three properties: (a) the position of the right-most 1 is the same in both strings, (b) all digits to the left of the right-most 1 are complements of each other in the two strings, and (c) there is at least one digit to the left of the right-most 1.

Consider two copies of the automaton in Figure 5.2 on page 99, one scanning the b-bit string representing d and the other scanning the b-bit string representing $2^b - d$, both right-to-left. Both automata reach state S_1 upon consuming the right-most 1. Thereafter, for every digit (0 or 1) processed as input by the first automaton, its complement is processed by the second. It is easy to see that the two automata are in diagonally opposite states at all times. Therefore, both traverse exactly the same number of thick edges when processing b bits. From Lemma 5.5.1, we conclude that if we were to run both automata for one more step, with an additional 0 as input, the number of thick edges traversed would be exactly the function G over the respective inputs. The only thick transition upon receiving 0 as input corresponds to $S_2 \xrightarrow{0} S_3$. Therefore, $G(d) = G(2^b - d)$ iff the automaton terminates in S_1 or S_3, and $G(d) = G(2^b - d) - 1$ iff the automaton terminates in S_0. Finally, $G(d) = G(2^b - d) + 1$ iff the automaton terminates in S_2. We are now left with the job of characterizing strings that terminate in the four states.

The automaton is in state S_2 when scanning a string right-to-left iff there exists a prefix of the string that satisfies the regular expression $1(01)^*1$. For $1 \le d < 2^b$, a prefix of the string representing d satisfies the regular expression $1(01)^*1$ iff $d > \lfloor 2^{b+1}/3 \rfloor$. Therefore, the automaton is in state S_2 iff $d > \lfloor 2^{b+1}/3 \rfloor$. Similarly, it can be shown that the automaton is in state S_0 iff $d \le \lfloor 2^b/3 \rfloor$. The automaton is in state S_1 or S_3 otherwise.

Thus $G(d) = G(2^b - d) + 1$ iff $d > \lfloor 2^{b+1}/3 \rfloor$, and $G(d) = G(2^b - d) - 1$ iff $d \le \lfloor 2^b/3 \rfloor$. Otherwise, $G(d) = G(2^b - d)$. □

Theorem 5.3. *An optimal solution to the Chord Routing Problem has path length*

$$\begin{array}{ll} 0 & \textit{if } d = 0 \\ G(d) & \textit{if } 1 \le d \le \lfloor 2^{b+1}/3 \rfloor \\ G(d) - 1 & \textit{otherwise} \end{array}$$

Proof. Follows from Lemma 5.5.2 and the definition of procedure OPT_ROUTE. □

In fact, Lemma 5.5.2 suggests an improvement to OPT_ROUTE, as shown in Figure 5.3 on the next page. This procedure is a complete characterization of the Chord Routing Problem and produces *all* possible optimal solutions for inputs b and d.

Theorem 5.4. RIGHT-TO-LEFT CHAINING *solves the Chord Routing Problem optimally. The diameter of Chord is* $\lfloor b/2 \rfloor$.

```
PROCEDURE Opt_Route(b, d)
    IF (d ≤ ⌊2^b/3⌋)
        ⟨d', d''⟩ ← Opt_Subtract(d)
        OUTPUT ⟨d', d''⟩
    ELSE IF (d > ⌊2^{b+1}/3⌋)
        ⟨d', d''⟩ ← Opt_Subtract(2^b − d)
        OUTPUT ⟨d'', d'⟩
    ELSE
                     ⎧ Opt_Subtract(2^b − d)
        ⟨d', d''⟩ ← ⎨ or
                     ⎩ Opt_Subtract(d)
        OUTPUT ⟨d', d''⟩
```

Figure 5.3: OPT_ROUTE(d) *is a non-deterministic procedure that solves the Chord Routing Problem.*

Proof. The path length of a route produced by RIGHT-TO-LEFT CHAINING for distance d is exactly equal to the number of thick edges traversed when scanning the binary representation of d right-to-left. As outlined in the proof of Lemma 5.5.1, RIGHT-TO-LEFT CHAINING traverses thick edges exactly $G(d)$ times iff $G(d) \leq \lfloor 2^{b+1}/3 \rfloor$. Otherwise, it traverses thick edges $G(d) - 1$ times. From Theorem 5.3, we see that this is optimal.

One of the strings that requires the maximum number of steps is $d = (01)^{b/2}$ when b is even and $d = (01)^{(b-1)/2}0$ when b is odd. For both cases, $G(d) = \lfloor b/2 \rfloor$. □

The proof of Theorem 5.4 sheds light on the reason for the automata for both RIGHT-TO-LEFT CHAINING and LEFT-TO-RIGHT BI-DIRECTIONAL GREEDY being identical in structure: the automaton is essentially a computation of G over the input string. The automaton remains the same irrespective of whether strings are scanned right-to-left or left-to-right.

5.5.4 Average Path Length

Theorem 5.3 characterizes the path length of optimal solutions for the Chord Routing Problem. Combining this theorem with Lemma 5.5.2, we see that the path length for $2^{b-1} < d < 2^b$ is simply $G(2^b - d)$. Therefore, the average path length over all distances $0 \leq d < 2^b$ is $[1 + 2\sum_{d=1}^{d=2^b-1} G(d)]/2^b$. (The additive constant 1 appears because the path length for $d = 0$ is zero, while it is 1 for $d = 2^{b-1}$.) It turns out that the computation of this average is greatly simplified if we analyze the automaton in Figure 5.2 on page 99.

We have already seen that the path length for a given distance d is just the number of thick edges traversed by the automaton when processing the b-bit binary representation of d. Computing the average path length simply requires us to run this automaton on all possible b-bit binary strings and compute the average number of thick edges traversed. This suggests an interesting approach for analysis.

We visualize the automaton as describing a 4-state Markov chain (each transition has probability 1/2) and we count the expected number of times a thick edge ($S_0 \xrightarrow{1} S_1$ or $S_2 \xrightarrow{0} S_3$) is traversed, in b steps. 1 in either d' or d'', thereby contributing to the sum $H(d') + H(d'')$. The stationary distribution for the 4-state Markov chain would be $[\ 1/3 \quad 1/6 \quad 1/3 \quad 1/6\ ]$. Assuming that the probability of being in a particular state converges to the stationary distribution very quickly (after a small number of bits have been processed by the automaton), the expected number of thick edges taken is roughly $(\frac{1}{2} \cdot \frac{1}{3} + \frac{1}{2} \cdot \frac{1}{3}) \cdot b = b/3$. Therefore, average path length should be $\approx b/3$ for large values of b. Our formal analysis validates this intuition by computing the exact probability distribution after ℓ bits are seen by the automaton, and computing the expected number of thick edges taken using exact probabilities.

For $\ell \geq 1$, let $A_{\ell,s}$ denote the fraction of binary strings of length exactly ℓ that cause the automaton to stop in state S_s on reading them. From Figure 5.2 on page 99, it follows that for $1 \leq \ell < b$, $A_\ell = A_0 P^\ell$, where matrix P and vectors A_ℓ and A_0 are defined as follows:

$$
\begin{aligned}
P &= \begin{bmatrix} 1/2 & 1/2 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 \\ 0 & 0 & 1/2 & 1/2 \\ 1/2 & 0 & 1/2 & 0 \end{bmatrix} \\
A_\ell &= [\ A_{\ell,0} \quad A_{\ell,1} \quad A_{\ell,2} \quad A_{\ell,3}\ ] \\
A_0 &= [\ 1 \quad 0 \quad 0 \quad 0\ ]
\end{aligned}
$$

Lemma 5.5.3. *For* $1 \leq \ell < b$,

$$
\begin{aligned}
A_{\ell,0} &= \lceil 2^\ell/3 \rceil / 2^\ell & \qquad A_{\ell,1} &= \lceil 2^\ell/6 \rceil / 2^\ell \\
A_{\ell,2} &= \lfloor 2^\ell/3 \rfloor / 2^\ell & \qquad A_{\ell,3} &= \lfloor 2^\ell/6 \rfloor / 2^\ell
\end{aligned}
$$

Proof. By induction. The base case ($\ell = 1$) is true because $A_1 = [\ 1/2 \quad 1/2 \quad 0 \quad 0\ ]$. The induction step follows from the fact that for $1 \leq \ell < b$,

$$
\begin{aligned}
A_{\ell+1,0} &= [A_{\ell,0} + A_{\ell,1} + A_{\ell,3}]/2 & \qquad A_{\ell+1,1} &= A_{\ell,0}/2 \\
A_{\ell+1,2} &= [A_{\ell,1} + A_{\ell,2} + A_{\ell,3}]/2 & \qquad A_{\ell+1,3} &= A_{\ell,2}/2
\end{aligned}
$$

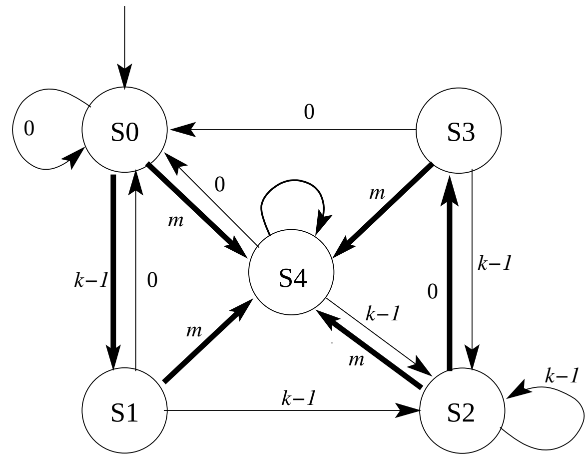


Figure 5.4: *State machine diagram for algorithm* Right-to-Left Chaining for Base-k.

□

Theorem 5.5. *The average path length for both* Left-to-Right Chaining *and* Right-to-Left Bi-directional Greedy *is* $b/3 + \Theta(1)$.

Proof. The average number of thick edges encountered for strings of length b is given by $\sum_{\ell=0}^{b-1}[A_{\ell,0}/2 + A_{\ell,2}/2]$. Plugging probabilities from Lemma 5.5.3 and using the identity $\lceil 2^\ell/3\rceil + \lfloor 2^\ell/3\rfloor = 2^{\ell+1}/3 + (-1)^\ell/3$ for $\ell \geq 0$, the sum simplifies to $\frac{b}{3} + \frac{1}{9}(1 - (\frac{-1}{2})^b)$. □

5.6 Chord in Base-k

In this Section, we study Chord in base-k, a natural generalization of Chord, similar to higher-base hypercube topologies. The higher base construction is of considerable practical interest because it offers routing in fewer steps than base-2 Chord when $k > 2$.

Definition (Chord in Base k). *Consider k^b nodes arranged clockwise in a circle with labels going from 0 to $k^b - 1$. An edge connects a pair of nodes iff they are separated by a clockwise or anti-clockwise distance of αk^β for some $0 < \alpha < b$ and $0 \leq \beta < b$.*

Right-to-Left Chaining for Base-k: The algorithm uses the automaton in Figure 5.4, which generalizes the one in Figure 5.2 on page 99. An extra state has been introduced to accommodate the middle digits, i.e., the digits other than 0 and $k - 1$. State

S_4 can actually be merged with state S_3 to obtain a smaller automaton. However, we keep the two states separate for clarity.

The input string d comprises of digits from the set $\{0, 1, \ldots k-1\}$ and is scanned from `right` to `left`. Symbol m is a short-hand for the "middle digits", i.e., the set $\{1, 2, \ldots k-2\}$. Each transition also produces a pair of digits as output, the first digit for d' and the second for d''. The start state is S_0. All thin edges produce $\langle 0, 0\rangle$ as output. All thick edges produce non-zero output and require *lookahead*:

$S_0 \overset{k-1}{\rightarrow} S_1$:
`if` (next digit is $k-1$)
`then output` $\langle 0, 1\rangle$
`else output` $\langle k-1, 0\rangle$

$S_2 \overset{0}{\rightarrow} S_3$:
`if` (next digit is $k-1$)
`then output` $\langle 0, k-1\rangle$
`else output` $\langle 1, 0\rangle$

$X \overset{m}{\rightarrow} S_4$:
`if` (next digit is $k-1$)
`then output` $\langle 0, m-1\rangle$
`else output` $\langle m, 0\rangle$

where X denotes any of the five states.

Theorem 5.6. *The average length of shortest paths for base-k Chord on k^b nodes is $\frac{k-1}{k+1}b + \Theta(1)$.*

Proof. The stationary distribution for the Markov chain associated with the automaton in Figure 5.4 on the preceding page is $[\ \frac{1}{k+1}\ \ \frac{1}{k(k+1)}\ \ \frac{1}{k+1}\ \ \frac{k-1}{k(k+1)}\ \ \frac{k-2}{k+1}\ ]$. A cost of 1 hop is paid every time a thick edge is traversed. Continuing in the same fashion as in the proof of Theorem 5.5, it can be shown that the average number of thick edge traversals is $\frac{k-1}{k+1}b + \Theta(1)$. □

5.7 The HyperSkewbe

We define the HyperSkewbe, a variant of the hypercube, as follows:

Definition (HyperSkewbe). *A HyperSkewbe of dimension k is defined on 2^k vertices, labeled 0 through $2^k - 1$. S_1 consists of two nodes with an edge between them. S_{k+1} is defined as follows: Consider one copy of S_k, and suffix all node identifiers with a 0 to create a graph $S_{k,0}$. Consider another copy of S_k and suffix all node identifiers with a 1 to create $S_{k,1}$. Now, create an edge between node x in $S_{k,0}$ and node $(x-1) \bmod 2^{k+1}$ in $S_{k,1}$. The resulting graph is S_{k+1}.*

We observe that the above recurrence is almost identical to the standard hypercube definition. The only difference is that node x in $S_{k,0}$ is attached to $(x-1) \bmod 2^{k+1}$ to create the HyperSkewbe, instead of to $x+1$, which would have resulted in the standard Hypercube. This small variation in the definition results in an intriguing structure which appears to have properties rather different from that of the Hypercube. For example, the diameter of the k-dimensional HyperSkewbe is smaller than k for all $k > 2$.

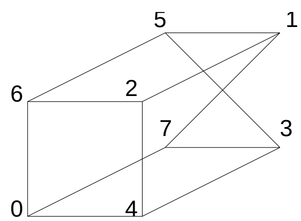


Figure 5.5: *3-dimensional HyperSkewbe on 8 nodes.*

Figure 5.5 shows a HyperSkewbe in three dimensions. The Chord network is simply the union of the hypercube of dimension k with the HyperSkewbe of dimension k, and it appears that it is the presence of this HyperSkewbe that makes the routing properties of Chord so different from that of the standard hypercube.

One algorithm for routing on the HyperSkewbe is to perform right-to-left bit-fixing, which results in an average path length of $b/2$ on the b-dimensional HyperSkewbe. One improvement to this routing algorithm is the following: Suppose we wish to route from node x to node y. We find the length of the route obtained by right-to-left bit-fixing to convert x to y. We also find the length of the route obtained by right-to-left bit-fixing converting y to x. In general, these two routes, and their lengths, are not identical. We can then choose the shorter of these two routes as the route from x to y. Experiments indicate that this idea improves upon simple right-to-left bit-fixing considerably. However, this algorithm is not optimal either. The characterization of optimal routes on the HyperSkewbe appears to be an interesting open problem.

5.8 Related Problems

1. A graph that is closely related to Chord is the hypercube:

   DEFINITION (Hypercube). *Consider an undirected graph on 2^b nodes, labeled with b-bit identifiers from 0 through $2^b - 1$. An edge (x, y) exists iff the labels of x and y differ at exactly one bit-position.*

   Each hypercube edge travels either a clockwise- or an anti-clockwise distance that is some power of two. Shortest paths in the hypercube are easy to characterize. For source s and destination d, a shortest path corresponds to fixing the bits of $s \oplus d$ in succession.

2. The following graph is obtained by replacing all occurrences of 2 by 3 in the definition of Chord:

   DEFINITION (Chord on 3^b Nodes). *Consider an undirected graph on 3^b nodes arranged in a circle. Nodes are labeled with b-bit identifiers from 0 through $3^b - 1$ going clockwise. An edge (x, y) exists iff x and y are 3^k positions apart on the circle for some $k \geq 0$, i.e., $|x - y|$ equals either 3^k or $3^b - 3^k$ for some $0 \leq k < b$.*

   Shortest paths in the graph above are much easier to characterize than shortest paths in Chord. The clockwise distance to some destination, x, can be identified by expressing it in ternary using the digits 0 and ± 1. Each $+1$ and -1 represents clockwise and anti-clockwise powers of three respectively. A shortest path corresponds to fixing the non-zero digits of x in any order.

5.9 Summary and Future Directions

We characterized shortest paths in Chord, a deterministic routing network defined for routing in DHTs. We showed how the problem can be reduced to the Binary Subtraction Problem: Given integer d, express it as $d = d' - d''$ such that the number of 1-bits in d' and d'' taken together is minimized. The average length of shortest paths in Chord is $b/3 + \Theta(1)$. The average was computed using an interesting technique: An algorithm for computing the shortest paths can be encoded as a finite-state automaton. Average path length is easy to compute if we treat the automaton as a Markov chain and compute its steady state distribution. Finally, Chord in base higher than two was analyzed and average length of shortest paths turns out to be $\frac{k-1}{k+1}b + \Theta(1)$.

Future research directions:

1. The characterization of shortest paths on the HyperSkewbe appears to be an interesting combinatorial problem.

2. Greedy routing on a circle, that forwards a message along that out-going link which minimizes the absolute distance remaining to the destination, is optimal for Chord. With $\Theta(\log n)$ links per node, Chord can route in $\Theta(\log n)$ steps. Is this tradeoff asymptotically optimal? We provide a partial answer to this question in Chapter 6, where we construct novel routing networks which can route in $\Theta(\log n / \log \log n)$ hops with greedy routing on the circle.

Chapter 6

Papillon: Greedy Routing on a Circle

In this Chapter, we study an interesting combinatorial problem. Consider n nodes labeled 0 through $n-1$, placed in a circle. GREEDY routing, formally defined below, is the strategy of forwarding a message along that out-going edge that minimizes the *distance* remaining to the destination:

> DEFINITION (Greedy Routing). *In graph (V, E) with distance function $\delta : V \times V \to \mathcal{R}^+$,* GREEDY *routing entails the following decision: Given a target node t, a node u with neighbors $N(u)$ forwards a message to its neighbor $v \in N(u)$ such that $\delta(v, t) = \min_{x \in N(u)} \delta(x, t)$.*

Two *natural* distance metrics over n nodes placed in a circle are the clockwise-distance and the absolute-distance between pairs of nodes:

$$\delta_{clockwise}(u, v) = \begin{cases} v - u & v \geq u \\ n + v - u & \text{otherwise} \end{cases}$$

$$\delta_{absolute}(u, v) = \begin{cases} \min\{v - u, n + u - v\} & v \geq u \\ \min\{u - v, n + v - u\} & \text{otherwise} \end{cases}$$

In this Chapter, we study the following related problems:

I. *Given integers d and Δ, what is the largest graph that satisfies two constraints: the out-degree of any node is at most d, and the length of the longest* GREEDY *route is at most Δ hops?*

II. *Given integers d and n, design a network in which each node has out-degree at most d such that the length of the longest* GREEDY *route is minimized.*

The motivation for these problems arises from our analysis of Chord in Chapter 5. In Chord, GREEDY routing with distance function $\delta_{absolute}$ routes along shortest paths (see Section 5.5.2 on page 100 for details). Chord has $\Theta(\log n)$ links per node. The longest GREEDY route has length $\Theta(\log n)$. The average length of GREEDY routes is also $\Theta(\log n)$. A question that arises is:

Given $\Theta(\log n)$ links per node, does GREEDY *routing with distance function $\delta_{absolute}$ (or for that matter, $\delta_{clockwise}$) entail $\Omega(\log n)$ hops in the worst case?*

We answer this question in the negative by constructing Papillon, a family of graphs in which each node has degree d, and GREEDY routes have length $O(\log n/\log d)$ in the worst-case. When $d = \Theta(\log n)$, GREEDY routes have length $O(\log n/\log\log n)$. Papillon is a variant of butterfly networks, and is asymptotically optimal in terms of the tradeoff between degree per node and worst-case GREEDY route length.

Summary of Results

In §6.1, we define Chord$_{clockwise}$ and Chord$_{absolute}$. Both are directed variants of Chord that permit efficient GREEDY routing with distance functions $\delta_{clockwise}$ and $\delta_{absolute}$ respectively. The motivation for defining the variants is to develop intuition for the butterfly-style constructions that we describe in subsequent Sections.

In §6.2, we construct two families of networks, the *Papillon*[†], having $n = d^{O(\Delta)} \cdot \Delta$ nodes. The two families correspond to distance functions $\delta_{clockwise}$ and $\delta_{absolute}$, respectively. The longest GREEDY route has length $\Delta = O(\log n/\log d)$ hops — this is asymptotically optimal, given n and d. Papillon is the first construction that achieves optimality for distance functions $\delta_{clockwise}$ and $\delta_{absolute}$.

In §6.3, further investigation of Papillon reveals the following interesting result: Papillon admits routing strategies that have shorter paths (in the constants) than GREEDY routing. This shows that GREEDY, though asymptotically optimal, does not route along shortest

[†]The constructions are variants of the well-known butterfly family, hence the name Papillon.

paths. We identify routing strategies that are superior to GREEDY. The improved strategies also guarantee uniform edge-congestion.

In §6.4, we present related work.

In §6.5, we summarize and outline future research directions.

6.1 Variants of Chord

Chord is an undirected graph that was defined at the beginning of Chapter 5. We now define two variants of Chord, each of which is a directed graph. Towards the end of this Section, we will show how these variants can be *morphed* into Papillon, which is discussed in §6.2.

DEFINITION ($\mathsf{Chord}_{clockwise}$). *Consider a graph on $n = \kappa^m$ nodes labeled from 0 to $n-1$, for any pair of integers $\kappa, m \geq 1$. Node u has κm edges with nodes $(u + i\kappa^j) \bmod n$, for all values of $i \in [1, \kappa]$ and $j \in [0, m-1]$.*

Theorem 6.1. GREEDY *routing with distance function $\delta_{clockwise}$ in Chord$_{clockwise}$ is along shortest paths. The longest route has m hops. Average route length is $(\kappa-1)m/\kappa$ hops.*

Proof. Write the clockwise distance remaining to the destination in base κ by using digits belonging to the set $[0, \kappa)$. Such a representation is unique for any integer. Forwarding a message according to GREEDY routing with distance function $\delta_{clockwise}$ amounts to replacing the most-significant non-zero digit in the representation by zero. Thus the longest GREEDY route has m hops. Since $(\kappa-1)m/\kappa$ digits are non-zero on average, we require as many hops on average. □

We define another variant of Chord that supports efficient GREEDY routing with distance function $\delta_{absolute}$.

DEFINITION ($\mathsf{Chord}_{absolute}$). *Consider a graph on $n = (2k+1)^m$ nodes labeled from 0 to $n-1$, for any pair of integers $k, m \geq 1$. Node u has $2km$ edges with nodes $(u + i(2k+1)^j) \bmod n$, for all values of $i \in [-k, +k]$, $i \neq 0$ and $j \in [0, m-1]$.*

In Chord, with $n = 2^b$ nodes, each node has $2b-1$ out-going links with nodes lying at clockwise distances $\langle \pm n/2, \pm n/4, \pm n/8, \ldots \pm 1 \rangle$ away from itself. Chord$_{absolute}$ with $k = 1$ has $n = 3^b$ nodes, where each node has $2b$ out-going links with nodes lying at clockwise distances $\langle \pm n/3, \pm n/9, \pm n/27, \ldots, \pm 1 \rangle$ away from itself. GREEDY routing with $\delta_{absolute}$ in

Chord$_{absolute}$ corresponds to writing the clockwise distance to the destination in ternary using the digits $\{-1, 0, +1\}$, and "fixing" the digits from left to right in succession to zeros. Only two-thirds of all digits are ± 1 on average. Thus average latency is $(2 \log_3 n)/3$. Generalizing this argument to higher values of k, we obtain the following theorem:

Theorem 6.2. GREEDY *routing with distance function $\delta_{absolute}$ in Chord$_{absolute}$ is along shortest paths. The longest route has m hops. Average route length is $2km/(2k + 1)$ hops.*

Proof. Write the clockwise distance remaining to the destination in base $2k + 1$ by using digits belonging to the set $[-k, k]$. Such a representation is unique for any integer. Forwarding a message according to GREEDY routing with distance function $\delta_{absolute}$ amounts to replacing the most-significant non-zero digit in the representation by zero. Thus the longest GREEDY route has m hops. Since $2km/(2k + 1)$ digits are non-zero on average, we require as many hops on average. $\square$

A Chord$_{absolute}$ network with $n = (2k+1)^m$ nodes and out-degree $2km$ per node, can be *morphed* into a butterfly-style network with $n = m(2k + 1)^m$ nodes and out-degree $2k + 1$ per node. We replace each Chord$_{absolute}$ node by a group of m nodes on the circle. These nodes are assigned *levels* 0 through $m - 1$, in the anti-clockwise direction. Any node in the new graph can be uniquely identified by its group (the label of the original Chord$_{absolute}$ node) and its level within the group. A node at level ℓ in group g makes $2k + 1$ connections with other nodes at level $(\ell + 1) \bmod m$ in groups $i(2k + 1)^g$, where $i \in [-k, +k]$. When $i = 0$, the connection is with the immediate successor. In the new network, we can route in at most $3m$ hops as follows: (a) route to a node at level $m - 1$ by following successor links, (b) route to the closest predecessor of the target at level $m - 1$, (c) follow successor links to reach the target. Curiously, GREEDY routing with distance function $\delta_{absolute}$ sometimes gets caught in infinite loops. For example, with $m \geq 5$, a message can never be sent from some node $\mathbf{x}$ at level 0 to its predecessor. With GREEDY routing, the message will get forwarded all the way to node $\mathbf{y}$ at level 0 in the next group. Node $\mathbf{y}$ will then forward the message back to $\mathbf{x}$, resulting in an infinite loop. This problem can successfully be "fixed" by incorporating an additional edge that connects a node to another node lying distance m away in the anti-clockwise direction (see the definition of $\mathcal{B}_{absolute}$ in §6.2 and Theorem 6.4).

A Chord$_{clockwise}$ network can be morphed into a butterfly network along the same lines as above.

6.2 Papillon

We construct two variants of butterfly networks, one each for distance-functions $\delta_{clockwise}$ and $\delta_{absolute}$. For convenience, $x \bmod n$ always represents an element lying in the range $[0, n-1]$ (even when x is negative, or greater than $n-1$).

DEFINITION (Papillon for $\delta_{clockwise}$). *$\mathcal{B}_{clockwise}(\kappa, m)$ is a directed graph, defined for any pair of integers $\kappa, m \geq 1$*

1. *The network has $n = \kappa^m m$ nodes labeled from 0 to $n-1$.*

2. *Let $\ell(u) \equiv (m-1) - (u \bmod m)$. Each node has κ links. For node u, these directed links are to nodes $(u+x) \bmod n$, where*

$$x \in \{1 + im\kappa^{\ell(u)} \mid i \in [0, \kappa - 1]\}$$

We denote the link with node $(u+1) \bmod n$ as u's "short link". *The other $\kappa - 1$ links are called u's* "long links".

DEFINITION (Papillon for $\delta_{absolute}$). *$\mathcal{B}_{absolute}(k, m)$ is a directed graph, defined for any pair of integers $k, m \geq 1$,*

1. *The network has $n = (2k+1)^m m$ nodes labeled from 0 to $n-1$.*

2. *Let $\ell(u) \equiv (m-1) - (u \bmod m)$. Each node has $2k+2$ out-going links. Node u makes $2k+1$ links with nodes $(u+x) \bmod n$, where*

$$x \in \{1 + im(2k+1)^{\ell(u)} \mid i \in [-k, +k]\}$$

Node u also makes an out-going link with node $(u+x) \bmod n$, where $x = -m+1$. We denote the link with node $(u+1) \bmod n$ as u's "short link". *The other $2k+1$ links are called u's* "long links".

In both $\mathcal{B}_{clockwise}$ and $\mathcal{B}_{absolute}$, all out-going links of node u are incident upon nodes with level $(\ell(u) - 1) \bmod m$. In $\mathcal{B}_{clockwise}$, GREEDY routing is guaranteed to take a finite number of hops – the existence of the short links ensures that each hop diminishes the remaining clockwise distance by at least one. In $\mathcal{B}_{absolute}$, not every GREEDY hop diminishes the remaining absolute distance, as will be clear in the proof of Theorem 6.4.

Theorem 6.3. GREEDY *routing in* $\mathcal{B}_{clockwise}$ *takes* $3m-2$ *hops in the worst-case. The average is less than* $2m-1$ *hops.*

Proof. For any node u, we define

$$\text{SPAN}(u) \equiv \{v \mid 0 \leq \delta_{clockwise}(u,v) < m\kappa^{\ell(u)+1}\}.$$

When $\ell(u) = m-1$, then SPAN(u) includes all the nodes. With t as the target node and u a current node, routing proceeds in three phases:

1. All out-going links of a node u are incident upon nodes at level $(\ell(u)-1) \bmod m$. Phase 1 terminates if the target is reached, or we reach a node at level $m-1$, whichever event happens first. This phase of routing takes at most $m-1$ hops (at most $\frac{m-1}{2}$ hops on average).

2. The following invariant is true: "$t \in$ SPAN(u) and $\delta_{clockwise}(u,t) \geq m$". GREEDY will forward the message to a node v such that $t \in$ SPAN(v) and $\ell(v) = \ell(u) - 1$. Eventually, a node u with $\ell(u) = 0$ will be reached. This node will forward the message to a node v with $\ell(v) = m-1$ such that $\delta_{clockwise}(v,t) < m$, thereby terminating this phase of routing. There are at most m hops in this phase.

3. The following invariant is true: "$t \in$ SPAN(u) and $\delta_{clockwise}(u,t) < m$". GREEDY will decrease the distance by exactly one, following the short-links. This phase takes at most $m-1$ hops (at most $\frac{m-1}{2}$ hops on average).

The worst-case route length is $3m-2$.

On average, routes are at most $2m-1$ hops long. □

Theorem 6.4. GREEDY *routing in* $\mathcal{B}_{absolute}$ *takes* $3m-2$ *hops in the worst-case. The average is less than* $2m-1$ *hops.*

Proof. For any node u, we define

$$\text{SPAN}(u) \equiv \{v \mid \delta_{absolute}(u,v) = |c + m\sum_{i=0}^{\ell(u)}(2k+1)^i d_i|,\ c \in [0, m-1],\ d_i \in [-k, +k]\,\}.$$

If $\ell(u) = m-1$, then SPAN(u) includes all n nodes. Routing proceeds in three phases:

1. All out-going links of a node u are incident upon nodes at level $(\ell(u) - 1) \bmod m$. Phase 1 terminates if the target is reached, or we reach a node at level $m-1$, whichever event happens first. This phase of routing takes at most $m - 1$ hops (at most $\frac{m-1}{2}$ hops on average).

2. The following invariant is true in this phase: "$t \in \mathrm{SPAN}(u)$ and $\delta_{absolute}(u, t) \geq m$", where u denotes the current node. At every step in this phase, if node u forwards the message to a node v using the rules for GREEDY routing, then it is guaranteed that $t \in \mathrm{SPAN}(v)$, and that $\ell(v) = \ell(u) - 1$. Phase 2 terminates if the target is reached, or if $\delta_{absolute}(u, t) < m$, or if the message is forwarded by a node u with $\ell(u) = 0$, whichever event happens first. When a node at level 0 forwards a message to some node v, then $\ell(v) = m - 1$ and it is guaranteed that $\delta_{absolute}(v, t) < m$, thereby terminating this phase of routing. There are at most m hops in this phase.

3. The following invariant is true in this phase: "$t \in \mathrm{SPAN}(u)$ and $\delta_{absolute}(u, t) < m$", where u denotes the current node. GREEDY is guaranteed to reach the destination in at most $m - 1$ hops (the existence of the "back edge" that connects node u to node $(u + 1 - m) \bmod n$ guarantees this). This phase takes at most $m - 1$ hops (at most $\frac{m-1}{2}$ hops on average).

Routes are at most $3m - 2$ hops in the worst-case.

The average length is at most $2m - 1$ hops. □

Routes in $\mathcal{B}_{clockwise}$ and $\mathcal{B}_{absolute}$ are at most $3m - 2$ hops, which is $O(\log(\kappa^m m)/\log \kappa)$ and $O(\log((2k+1)^m m)/\log(2k+2))$, respectively. Given degree d and diameter Δ, the size of Papillon is $n = 2^{O(\Delta)}\Delta$ nodes. Given degree d and network size n, the longest route has length $\Delta = O(\log n/\log d)$.

6.3 Improved Routing Algorithms for Papillon

GREEDY routing does not route along shortest-paths in $\mathcal{B}_{clockwise}$ and $\mathcal{B}_{absolute}$. We demonstrate this constructively below, where we study a routing strategy called HYPERCUBIC-ROUTING which achieves shorter path lengths than GREEDY.

Hypercubic Routing

Theorem 6.5. *There exists a routing strategy for $\mathcal{B}_{clockwise}$ in which routes take $2m-1$ hops in the worst-case. The average is at most $1.5m$ hops.*

Proof. Consider the following HYPERCUBIC-ROUTING algorithm on $\mathcal{B}_{clockwise}$. Let s be the source node, t the target, and let $dist = \delta_{clockwise}(s,t) = c + m + m\sum_{i=0}^{i=m-1} \kappa^i d_i$ with $0 \le c < m$ and $0 \le d_i < \kappa$ ($dist$ has exactly one such representation, unless $dist \le m$ in which case routing takes $< m$ hops).

Phase I: Follow the short-links to "fix" the c-value to zero. This takes at most $m-1$ hops (at most $0.5m$ hops on average).

Phase II: In exactly m hops, "fix" the d_i's in succession to make them all zeros: When the current node is u, we fix $d_{\ell(u)}$ to zero by following the appropriate long-link, i.e., by shrinking the clockwise distance by $d_{\ell(u)}\kappa^{\ell(u)}m+1$. The new node v satisfies $\ell(v) = (\ell(u) + m - 1)(\text{mod } m)$. When each d_i is zero, we have reached the target.

Overall, the worst-case route length is $2m-1$. Average route length is at most $1.5m$. ☐

Theorem 6.6. *There exists a routing strategy for $\mathcal{B}_{absolute}$ in which routes take $2m-1$ hops in the worst-case. The average is at most $1.5m$ hops.*

Proof. Consider the following HYPERCUBIC-ROUTING algorithm on $\mathcal{B}_{absolute}$. Let s be the source node, t the target, and let $dist = c + m + m\sum_{i=0}^{i=m-1}(2k+1)^i d_i$, where $0 \le c < m$ and $-k \le d_i \le k$ ($dist$ has exactly one such representation, unless $dist < m$ in which case routing takes fewer than m hops). Note that $|dist| = \delta_{absolute}(s,t)$.

Phase I: Follow the short-links in the clockwise direction, to reach a node s' such that $\ell(s') = \ell(t)$. This takes at most $m-1$ hops (at most $0.5m$ hops on average). The remaining distance can be expressed as $m + m\sum_{i=0}^{i=m-1}(2k+1)^i d_i$ with $-k \le d_i \le k$.

Phase II: In exactly m hops, "fix" the d_i's in succession to make them all zeros: When the current node is u, we fix $d_{\ell(u)}$ to zero by following the appropriate long-link, i.e., by traveling distance $1 + d_{\ell(u)}(2k+1)^{\ell(u)}m$ along the circle (this distance is positive or negative, depending upon the sign of $d_{\ell(u)}$). The new node v satisfies $\ell(v) = (\ell(u) - 1)(\text{mod } m)$. When each d_i is zero, we have reached the target.

Overall, the worst-case route length is $2m-1$. Average route length is at most $1.5m$. ☐

Note that the edges that connect node u to node $(u + 1 - m) \bmod n$ are redundant for

HYPERCUBIC-ROUTING since they are never used. However, these edges play a crucial role in GREEDY routing in $\mathcal{B}_{absolute}$ (to guide the message to the target in Phase 3).

Congestion-Free Routing

Theorems 6.5 and 6.6 prove that GREEDY routing is sub-optimal in the constants. Still, HYPERCUBIC-ROUTING, as described above, causes *edge-congestion* because short-links are used more often than long-links. Let π denote the ratio of maximum and minimum loads on edges caused by all $\binom{n}{2}$ pairwise routes. HYPERCUBIC-ROUTING for $\mathcal{B}_{clockwise}$ consists of two phases (see Proof of Theorem 6.5). The load due to Phase II is uniform – all edges (both short-links and long-links) are used equally. However, Phase I uses only short-links, due to which $\pi \neq 1$. We now modify the routing scheme slightly to obtain $\pi = 1$ for both $\mathcal{B}_{clockwise}$ and $\mathcal{B}_{absolute}$.

Theorem 6.7. *There exists a congestion-free routing strategy in $\mathcal{B}_{clockwise}$ that takes $2m-1$ hops in the worst-case and at most $1.5m$ hops on average, in which $\pi = 1$.*

Proof. The theorem is proved constructively, by building a new routing strategy called CONGESTION-FREE. This routing strategy is exactly the same as HYPERCUBIC-ROUTING, with a small change.

Let s be the source node, t the target. Let $c = (t + m - s) \bmod m$, the difference in levels between $\ell(s)$ and $\ell(t)$.

Phase I: For c steps, follow any out-going link, chosen uniformly at random. We thus reach a node s' such that $\ell(s') = \ell(t)$.

Phase II: The remaining distance is $dist = \delta_{clockwise}(s', t) = m + m \sum_{i=0}^{i=m-1} \kappa^i d_i$ with $0 \leq d_i < \kappa$. Continue with Phase II of the HYPERCUBIC-ROUTING algorithm for $\mathcal{B}_{clockwise}$ (see Theorem 6.5).

It is easy to see that in this case, all outgoing links (short- and long-) are used with equal probability along the route. Hence, $\pi = 1$. □

Theorem 6.8. *There exists a congestion-free routing strategy in $\mathcal{B}_{absolute}$ that takes $2m-1$ hops in the worst-case and at most $1.5m$ hops on average, in which $\pi = 1$.*

Proof. We will ignore the edges that connect node u to node $(u + 1 - m) \bmod n$ (recall that these edges are not used in HYPERCUBIC-ROUTING described in Theorem 6.6). We will ensure $\pi = 1$ for the remainder of the edges.

CONGESTION-FREE routing follows the same idea as that for $\mathcal{B}_{clockwise}$ (Theorem 6.7): Let s be the source node, t the target. Let $c = (t + m - s) \bmod m$, the difference in levels between $\ell(s)$ and $\ell(t)$. In Phase I, for c steps, we follow any out-going link, chosen uniformly at random. We thus reach a node s' such that $\ell(s') = \ell(t)$. In Phase II, we continue as per Phase II of the HYPERCUBIC-ROUTING algorithm for $\mathcal{B}_{absolute}$ (Theorem 6.6).

An alternate CONGESTION-FREE routing algorithm for $\mathcal{B}_{absolute}$ that routes deterministically is based upon the following idea: We express any integer $a \in [-k, +k]$ as the sum of two integers: $a' = \lfloor (k + a)/2 \rfloor$ and $a'' = -\lfloor (k - a)/2 \rfloor$. It is easy to verify that $a = a' + a''$. Now if we list all pairs $\langle a', a'' \rangle$ for $a \in [-k, +k]$, then each integer in the range $[-k, +k]$ appears exactly twice as a member of some pair.

Let s be the source node, t the target. Let $c = (t + m - s) \bmod m$, the difference in levels between $\ell(s)$ and $\ell(t)$. The remaining distance is $dist = c + m + m \sum_{i=0}^{i=m-1} (2k+1)^i d_i$ with $-k \le d_i \le k$ (there is a unique way to represent $dist$ in this fashion).

Phase I: For c steps, if the current node is u, then we follow the edge corresponding to $d'_{\ell(u)}$, i.e., the edge that covers distance $1 + m d'_{\ell(u)} (2k + 1)^{\ell(u)}$ (in the clockwise or the anti-clockwise direction, depending upon the sign of $d'_{\ell(u)}$). At the end of this phase, we reach a node s' such that $\ell(s') = \ell(t)$.

Phase II: Continue with Phase II of the HYPERCUBIC-ROUTING algorithm for $\mathcal{B}_{absolute}$ (Theorem 6.6), for exactly m steps.

Due to the decomposition of integers in $[-k, +k]$ into pairs, as defined above, all outgoing links (short- and long-) are used equally. Hence, $\pi = 1$. $\square$

Notes: In the context of the current Internet, out-going links correspond to full-duplex TCP connections. Therefore, the undirected graph corresponding to $\mathcal{B}_{absolute}$ is of interest. In this undirected graph, it is possible to devise congestion-free routing with $\pi = 1$, maximum path length $m + \lfloor m/2 \rfloor$ and average route-length at most $1.25m$. This is achieved by making at most $\lfloor m/2 \rfloor$ initial random steps either in the down or the up direction, whichever gets to a node with level $\ell(t)$ faster.

6.4 Related Problems

The problem of Greedy Routing on a Circle has relationships with the following problems:

1. Efficient graph constructions are known that support GREEDY routing with distance function other than $\delta_{clockwise}$ and $\delta_{absolute}$. For example, in hypercubes, shortest paths between node-pairs correspond to GREEDY routing with distance function $\delta(u,v) \equiv u \oplus v$, the number of bit-positions where u and v differ. For de Bruijn networks, the traditional routing algorithm (which routes almost always along shortest paths) corresponds to GREEDY routing with $\delta(u,v)$ defined as the longest suffix of u that is also the prefix of v. For a 2D grid, shortest paths correspond to GREEDY routing with $\delta(u,v)$ defined as the Manhattan distance between nodes u and v. However, the problem of GREEDY routing with distance functions $\delta_{clockwise}$ and $\delta_{absolute}$ has not been studied before.

2. GREEDY routing with distance function $\delta_{absolute}$ has been studied for Chord (see Section 5.5.2 on page 100). Chord has 2^b nodes, with out-degree $2b-1$ per node. The longest GREEDY route takes $\lfloor b/2 \rfloor$ hops. In terms of d and Δ, the largest-sized Chord network has $n = 2^{2\Delta+1}$ nodes. Moreover, d and Δ cannot be chosen independently – they are functionally related. Both d and Δ are $\Theta(\log n)$.

   GREEDY routing with distance function $\delta_{clockwise}$ for Chord$_{clockwise}$ was studied in §6.1. In the same Section, GREEDY routing with distance function $\delta_{absolute}$ was studied for Chord$_{absolute}$.

3. Xu *et al* [XKY03] have studied GREEDY routing with distance function $\delta_{clockwise}$ over *uniform* graph topologies. A graph over n nodes placed in a circle is said to be uniform if the set of clockwise offsets of out-going links is identical for all nodes. Chord is an example of a uniform graph. Xu *et al* show that for any uniform graph with $O(\log n)$ links per node, GREEDY routing with distance function $\delta_{clockwise}$ necessitates $\Omega(\log n)$ hops in the worst-case.

   Cordasco *et al* [CGH+04] have shown that GREEDY routing with distance function $\delta_{clockwise}$ in a uniform graph over n nodes satisfies the inequality $n \leq F(d+\Delta+1)$, where d denotes the out-degree of each node, Δ is the length of the longest GREEDY path, and $F(k)$ denotes the k^{th} Fibonacci number. It is well-known that $F(k) = [\phi^k/\sqrt{5}]$, where $\phi = 1.618\ldots$ is the Golden ratio and $[x]$ denotes the integer closest to real number x. It follows that $1.44 \log_2 n \leq d + \Delta + 1$. Cordasco *et al* show that the inequality is strict if $|d - \Delta| > 1$. For $|d - \Delta| \leq 1$, they construct uniform graphs based upon Fibonacci numbers which achieve an optimal tradeoff between d and Δ.

Papillon is a non-uniform graph since the set of clockwise offsets of out-going links is different for different nodes.

4. The Degree-Diameter Problem, studied in extremal graph theory, seeks to identify the largest graph with diameter Δ, with each node having out-degree at most d (see Delorme [D04] for a survey). The best constructions for large Δ tend to be sophisticated [BDQ92, CG92, E01b]. A well-known upper bound is $N(d, \Delta) = 1 + d + d^2 + \cdots + d^\Delta = \frac{d^{\Delta+1}-1}{d-1}$, also known as the Moore bound. A general lower bound is $d^\Delta + d^{\Delta-1}$, achieved by Kautz digraphs [K68, K69], which are slightly superior to de Bruijn graphs [dB46] whose size is only d^Δ. A consequence of these results is that it is indeed possible to route in $O(\log n / \log d)$ hops in the worst-case with d out-going links per node. Whether GREEDY routes with distance functions $\delta_{clockwise}$ or $\delta_{absolute}$ can achieve the same bound, is the question we addressed in this Chapter.

5. GREEDY routing with distance function $\delta_{clockwise}$ has recently been studied for certain classes of *random graphs* (see Chapters 7 and 8 for more details).

   A brief summary of results pertaining to GREEDY routing in random graphs is as follows: Kleinberg [K00] showed that if each node chooses one out-going link from a certain probability distribution function, the length of GREEDY routes is $O(\log^2 n)$ in the worst case. Barrière *et al* [BFKK01] established a matching lower bound of $\Omega(\log^2 n)$ hops. With d out-going links per node, GREEDY routing results in routes of length $O(\frac{1}{d}\log^2 n)$ hops (see reference [MBR03] or Chapter 7 for details). Aspnes *et al* [ADS02] proved that any symmetric, randomized degree-d network has $\Omega(\frac{\log^2 n}{d \log\log n})$ GREEDY routing complexity.

   A variant of GREEDY routing is "GREEDY with 1-LOOKAHEAD" – the idea is to take the neighbor's neighbors of a node into account and to optimize for two hops at once. In reference [MNW04], GREEDY routing with/without 1-LOOKAHEAD was analyzed for a variety of graphs: randomized-Chord [GGG$^+$03, ZGG03], randomized-hypercubes [GGG$^+$03], Symphony [MBR03], skip-graphs [AS03] and SkipNet [HJS$^+$03]. Each of these networks has $O(\log n)$ out-going links per node. It was established that GREEDY routing requires $\Omega(\log n)$ hops on average. However, aided by 1-LOOKAHEAD, the length of routes shrinks to $\Theta(\log n / \log\log n)$ hops, which is optimal.

   The above results motivate the following question: Is it possible to construct a graph in which each node has degree d and in which GREEDY *without* 1-LOOKAHEAD has routes of length $\Theta(\log n/\log d)$ in the worst case? Papillon provides an answer in the affirmative.

Butterfly networks have been used in the context of routing networks for DHTs as follows:

1. Deterministic butterflies have been proposed for DHT routing by Xu *et al* [XKY03], who subsequently developed their ideas into Ulysses [KMXY03]. Papillon for distance function $\delta_{clockwise}$ has structural similarities with Ulysses – both are butterfly-based networks. The key differences are as follows: (a) Ulysses does not use $\delta_{absolute}$ as its distance function, (b) Ulysses does not use GREEDY routing, and (c) Ulysses uses more links than Papillon for distance function $\delta_{clockwise}$ – additional links have been introduced to ameliorate non-uniform edge congestion caused by Ulysses' routing algorithm. In contrast, the CONGESTION-FREE routing algorithm developed in §6.3 obviates the need for any additional links in Papillon (see Theorem 6.7).

2. Viceroy [MNR02] is a *randomized* butterfly network which routes in $O(\log n)$ hops in expectation with $\Theta(1)$ links per node. Mariposa (see reference [M03] or Chapter 9) improves upon Viceroy by providing routes of length $O(\log n/\log d)$ in the worst-case, with d out-going links per node. Viceroy and Mariposa are different from other randomized networks in terms of their design philosophy (see Chapter 9 for details).

6.5 Summary and Future Directions

Papillon is a variant of multi-butterfly networks which supports asymptotically optimal GREEDY routes of length $O(\log n/\log d)$ with distance functions $\delta_{clockwise}$ and $\delta_{absolute}$, when each node makes d out-going links, in an n-node network. Papillon is the first construction with this property.

Some questions that remain unanswered:

1. *Is it possible to devise graphs in which* GREEDY *routes with distance function* $\delta_{clockwise}$ *and* $\delta_{absolute}$ *are along shortest-paths?* As Theorems 6.5 and 6.6 illustrate, GREEDY routing on Papillon do not route along shortest-paths. Is this property inherent in GREEDY routes?

2. *What is the upper-bound for the Problem of Greedy Routing on the Circle?* Papillon furnishes a lower-bound, which is asymptotically optimal. However, constructing the

largest-possible graph with degree d and diameter Δ, is still an interesting combinatorial problem.

Chapter 7

Symphony: Routing in a Small-World

It's a small world, but I wouldn't want to have to paint it.

STEVEN WRIGHT (1955 –)

In this Chapter, we present the design of Symphony, a *randomized* family of routing networks for the module "Choice of Long-Distance Links" in Dipsea (see Figure 1.1 on page 2 for a block-diagram of its architecture).

7.1 Introduction

The *small world phenomenon* was discovered by Milgram [M67] in a celebrated experiment that demonstrated that pairs of people in a society were connected by short chains of acquaintances. Milgram also discovered that people were actually able to *route* letters to unknown persons in a few hops by forwarding them through acquaintances. To model the small world phenomenon, Kleinberg [K00] recently constructed a 2D grid where every node maintains four links to each of its closest neighbors and *one* long-distance link to a node chosen from a harmonic probability distribution. In the resulting network, a message can be routed from any node to another by *greedy routing* in $O(\log^2 n)$ hops on average.

In this Chapter, we present Symphony, an adaptation of Kleinberg's construction for maintaining a randomized routing network over a *dynamic* set of nodes lying on a circle.

Each node makes a *short-distance* link with its successor along the circle, and multiple *long-distance* links with other nodes. In contrast, Kleinberg's construction was over a static set of n^2 nodes arranged in a 2D grid. The issues we address are: *How is the network maintained in the face of arrivals and departures of nodes? How good is greedy routing with $k > 1$ long-distance links per node? How does the construction compare with other deterministic and randomized routing networks?*

The emphasis in this Chapter is on engineering a practical design – we provide extensive experimental results to justify our design decisions. Further theoretical results pertaining to Symphony and other randomized routing networks are available in Chapter 8.

Summary of Results

In §7.2, we describe three distributions of node IDs: Regular, Balanced and Random.
In §7.3, we study Symphony over a Regular distributions of IDs.
In §7.4, we study Symphony over a Balanced distributions of IDs.
In §7.5, we study Symphony over a Random distributions of IDs.
In §7.6, we provide experimental results.
In §7.7, we compare Symphony with other routing networks.
In §7.8, we summarize and present future work.

7.2 ID Distributions

We will describe Symphony over three different distributions of nodes lying on the circumference of a circle. It is convenient to imagine the perimeter of the circle as the unit interval $\mathcal{I} = [0, 1)$, "wrapped around". Each node is assigned an ID which is a fraction in $\mathcal{I}$.

- ⋆ Regular Distribution: The n nodes occupy positions corresponding to the corners of a regular n-gon circumscribed by the circle.
- ⋆ Random Distribution: Each node has chosen its position independently and uniformly at random on the circumference of the circle.
- ⋆ Balanced Distribution: Node IDs correspond to leaf nodes of a binary tree whose leaves belong to at most three different levels: $[\log_2 n]$ and $[\log_2 n] \pm 1$, where n is the number of leaves, and $[x]$ denotes the integer closest to real number x. The ID of a leaf is simply the sequence of zeros and ones encountered along the path from the root to that leaf, treated as a fraction in $\mathcal{I}$.

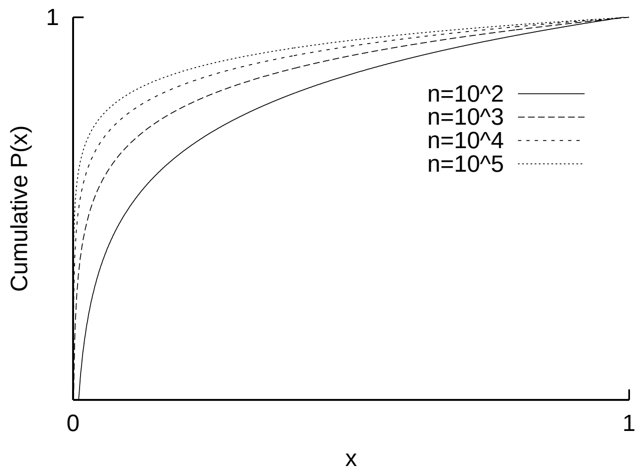


Figure 7.1: *Cumulative probability distribution function* $P_n(x) \equiv \int_{1/n}^{x} p_n(x)\mathrm{dx}$ *for different* n.

Balanced distribution of IDs results from ID management algorithms that we discussed in Chapters 2 and 3.

7.3 Symphony over Regular Distribution of IDs

We define $p_n(x)$, a probability distribution function, as $p_n(x) = 1/(x \ln n)$ for $x \in [1/n, 1]$. It is easy to see that $\int_{1/n}^{1} p_n(x)\mathrm{dx} = 1$. See Figure 7.1 for a plot of $p_n(x)$ for different values of n. Drawing from p_n corresponds to a simple C expression: `exp (log(n) * (drand48() - 1.0))`, where `drand48()` produces a random number between 0 and 1.

A node in Symphony maintains *two short links* with its immediate neighbors along the circle. In addition, a node maintains $k \geq 1$ *long distance links*. For each such link, a node first draws a random number $x \in I$ from $p_n(x)$ and makes a link with the node lying clockwise distance $\lceil x \rceil$ away from itself. The resulting randomized network supports efficient clockwise-greedy routing, defined as follows:

> Definition (Clockwise-Greedy Routing with Uni-directional Links). *A node forwards a message for* $x \in \mathcal{I}$ *along that out-going link (short or long) that minimizes the* <u>*clockwise distance*</u> *to* x.

Kleinberg [K00] analyzed the case $k = 1$. He showed that the expected route length for clockwise-greedy routing is $O(\log^2 n)$ hops. We now show that in general, for $k \leq \log_2 n$, the expected route length is $O(\frac{1}{k}\log^2 n)$ hops.

Theorem 7.1. *With Regular distribution of IDs, the expected path length with Clockwise-Greedy Routing using Uni-directional Links is $O(\frac{1}{k}\log^2 n)$ hops, with $k \leq \log_2 n$ long-distance links per node in an n-node network.*

Proof. Recall that $p_n(x) = 1/(x \log n)$ for $x \in [1/n, 1]$. Let p_{half} denote the probability of drawing a value from $[z/2, z]$. For any $z \in [2/n, 1]$, $p_{\text{half}} = \int_{z/2}^{z} p_n(x)dx = 1/\log_2 n$, which is independent of z. The significance of p_{half}: regardless of the current clockwise-distance to the destination, it is the probability that any single long-distance link will cut the distance by at least half. The number of links to consider before the current distance diminishes by at least half follows a geometric distribution with mean $1/p_{\text{half}} = \log_2 n$. With k links per node, the expected number of nodes to consider before the current distance is at least halved is $\lceil (\log_2 n)/k \rceil$, which is less than $(2\log_2 n)/k$ for $k \leq \log_2 n$. The maximum number of times the original distance could possibly be halved before it is less than $2/n$ is $\log_2(n/2)$. Thus the expected length of route is at most $2(\log_2 n)(\log_2(n/2))/k = O(\frac{1}{k}\log^2 n)$ hops.

An alternative proof for the $O(\frac{1}{k}\log^2 n)$ bound is as follows. Consider the following process: A particle starts at position p where $p > 1$ is an integer. At successive time-steps, the particle moves to position $p - X$, where X is a random variable ranging over the integers $1, \ldots, p-1$. The process terminates if the particle reaches position $p = 1$. Let $T(p)$ denote the number of steps required for the process to terminate. Theorem 1.3 in Motwani and Raghavan [MR95] establishes that if $\mathbf{E}X \geq g(p)$, where $g(p)$ is a monotone non-decreasing function, then $\mathbf{E}T(p) \leq \int_1^p \mathrm{dp}/g(p)$. The probability that the remaining distance is at least halved is given by $p_{\text{half}} = \int_{z/2}^{z} p_n(x)\mathrm{dx} = 1/\log_2 n$. With k long-distance links per node, the probability that at least one of these links diminishes the remaining distance by at least half is $1 - (1 - 1/\log_2 n)^k \geq k/(2\log_2 n)$. Therefore, $\mathbf{E}X \geq \frac{X}{2}\frac{k}{2\log_2 n}$. Using Theorem 1.3 in reference [MR95], we obtain that $\mathbf{E}T(n) \leq \int_1^n \frac{2}{X}\frac{2\log_2 n}{k}\mathrm{dX} = O(\frac{1}{k}\log^2 n)$. □

Notes:

1. Barrière *et al* [BFKK01] established that for $k = 1$, clockwise-greedy routing requires $\Omega(\log^2 n)$ hops on average, thereby furnishing a lower-bound for Kleinberg's $O(\log^2 n)$ bound. In Chapter 8, we establish that with $k \leq \log_2 n$ links per node, clockwise-greedy routing requires $\Omega(\frac{1}{k}\log^2 n)$ hops on average.

2. The proof of Theorem 7.1 is simpler than Kleinberg's original proof [K00].

3. See Martel and Nguyen [MN04] for an alternative proof of the $\Theta(\log^2 n)$ bound for $k = 1$.

4. It is important that links be chosen following a harmonic distribution. If each of the k long-distance links is made with a node whose clockwise distance is chosen uniformly at random from the interval $[0, 1)$, then using the proof technique in [K00], it can be shown that the average latency is $\Omega(\sqrt{n/k})$ hops. Figure 7.12 on page 139 presents a graphical illustration of this observation.

When each node makes k links with other nodes, the expected number of *incoming links* per node is also k. In practice, links are TCP connections which are full duplex, *i.e.,* traffic can be sent along both directions. One way to leverage incoming links is to treat them as additional long distance links and continue to use clockwise-greedy routing. However, this helps reduce average latency only marginally. Much more benefit can be obtained by exploiting the following insight: The distribution of the source ID of an incoming link corresponds roughly to p_n but in the anti-clockwise direction. The observation that a node has exactly k clockwise and roughly k anticlockwise long distance links motivates the following routing protocol:

DEFINITION (Absolute-Greedy Routing with Bi-directional Links). *A node forwards a message for $x \in \mathcal{I}$ along that link (incoming or out-going) that minimizes the <u>absolute distance</u> to x along the circle.*

7.4 Symphony over **Balanced Distribution** of IDs

In Chapter 4, we showed how emulation of families of networks defined on successive powers of two could be carried out for Balanced distribution of IDs. To use that emulation scheme, all we are required to do is to define Symphony over successive powers of two, assuming a Regular distribution of IDs for each power. The number of long-distance links can be a fixed number, say $k = 20$, or be made a function of the number of nodes *e.g.,* $k = \log_2 n$, where n is the power of two under consideration.

Using the emulation scheme in Section 4.1, a node with an ℓ-bit ID makes three sets of long-distance links: using its top $(\ell - 2)$-bits, its top $(\ell - 3)$ bits and its top $(\ell - 4)$-bits respectively. In the case of Symphony, these correspond to setting up long-distance links with three different probability distributions: One set of links with $p_n(x)$, another with

$p_{2n}(x)$ and a third with $p_{4n}(x)$, where $n = 2^{\ell-2}$. An interesting observation is that the set of links with $p_{4n}(x)$ can be "re-used" when establishing the set of links with $p_{2n}(x)$, which in turn can be "re-used" when establishing the set of links with $p_n(x)$. Consider the distribution $p_{2n}(x)$ defined over $[1/2n, 1]$. The probability that $p_{2n}(x)$ lies between $1/2n$ and $1/n$ is $\alpha = 1/\log_2 n$. Let $p'_{2n}(x)$ denote the probability distribution obtained by restricting $p_{2n}(x)$ to the range $[1/n, 1]$ and scaling it by $\frac{1}{1-\alpha}$. Then $\int_{k/n}^{(k+1)/n} p_n(x)\mathrm{dx} = \int_{k/n}^{(k+1)/n} p'_{2n}(x)\mathrm{dx}$, for any $1 \le k < n$. If k links are being established when the network size is $2n$, the expected number of links that cannot be "re-used" is $k\alpha$. For example, if $k = \log_2 2n$, then only one link cannot be "re-used" – the rest are common.

Theorem 7.2. *With Balanced distribution of IDs, the expected path length with Clockwise-Greedy Routing using Uni-directional Links is $O(\frac{1}{k}\log^2 n)$ hops, with $k \le \log_2 n$ long-distance links per node in an n-node network.*

Proof. Follows from Theorem 7.1 and the emulation scheme described in Section 4.1. □

7.5 Symphony over Random Distribution of IDs

In Chapter 4 (see §4.3), we described a general scheme for emulating families of routing networks defined over successive powers of two, assuming Random distribution of IDs. In this Section, we describe an alternative emulation scheme, tailored specifically for Symphony. The motivation for doing so is that the scheme described in this Section is much simpler than the scheme in Section 4.3.

7.5.1 Estimation Protocol

When the distribution of IDs is Random, each node has to first estimate the current number of nodes in the system. In a peer-to-peer system, it is difficult for all nodes to agree on the exact value of current number of participants n, especially in the face of frequent arrivals and departures of nodes. However, we can design an Estimation Protocol based on the following insight: Let X_s denote the sum of arc lengths managed by any set of s distinct nodes. Then $\frac{s}{X_s}$ is an unbiased estimator for n. The estimate improves as s increases. Our experiments show that $s = 3$ is good enough in practice. A node estimates n by using the length of the arc it partitions and its two neighboring arcs. These three arc-lengths are readily available at no extra cost from the two nodes between which x inserts itself in the ring. In Section 7.6.1, we show that the impact of increasing s on average route lengths is insignificant.

The idea of estimating n by using $s = 1$ was first proposed in Viceroy [MNR02] where it was shown that $\frac{1}{2}\log n < \lfloor \log \tilde{n} \rfloor < 3 \log n$ w.h.p. Since then, stronger results have been derived: Naor and Wieder [NW03] showed that the longest arc-length is $\Theta(\frac{\log n}{n})$ w.h.p. King and Saia [KS04] established that the smallest arc-length is $\Theta(\frac{1}{n^2})$ w.h.p. Using these results, we can assert that $\log n - \log\log n + \Theta(1) < \log \tilde{n} < 2\log n + \Theta(1)$, w.h.p.

7.5.2 Link Establishment

Let $\tilde{n}$ denote the estimate of n maintained by a node. Let $p_{\tilde{n}}(x)$ denote the probability distribution function $p_{\tilde{n}}(x) = 1/(x \log \tilde{n})$ for $x \in [1/\tilde{n}, 1]$. A node maintains *two short links* with its immediate neighbors. In addition, a node maintains $k \geq 1$ *long-distance links* as follows. For any link, a node first draws a random number $x \in I$ from $p_{\tilde{n}}(x)$. It then contacts the manager of the point x away from itself in the clockwise direction by following a Routing Protocol which we describe in Section 7.5.3. Finally, it attempts to establish a link with the manager it just identified.

We ensure that the number of incoming links per node is bounded by placing an upper limit of $2k$ incoming links per node. Once the limit is reached, all subsequent requests to establish a link with this node are rejected. The requesting node then makes another attempt by re-sampling from its probability distribution function. As a practical matter, an upper bound is placed on the number of such attempts, before a node gives up. We also ensure that a node does not establish multiple links with another node.

Choice of k: The number of out-going links established by each node is a design parameter. We could set k to a fixed value, say 20, or make it a function of $\tilde{n}$, the estimate of n maintained by a node. A reasonable design choice is $k = \lceil \log_2 \tilde{n} \rceil$. We experimentally show that as few as four long distance links are sufficient for small average route lengths.

7.5.3 Greedy Routing Protocols

We will use the same two routing protocols that we defined in 7.3, namely Clockwise-Greedy Routing with Uni-directional Links and Absolute-Greedy Routing with Bi-Directional Links.

Theorem 7.3. *With Random distribution of IDs, the expected path length with Clockwise-Greedy Routing using Uni-directional Links is $O(\frac{1}{k}\log^2 n)$ hops, with $k \leq \log_2 n$ long-distance* *links per node in an n-node network.*

Proof. We sketch the proof assuming every attempted long-distance link is successful. As noted above in Section 7.5.2, some of these connections in fact are rejected because the intended target of the link is "saturated" with $2k$ incoming links.

A node with estimate $\tilde{n}$ uses the probability distribution function $p_{\tilde{n}}(x) = 1/(x \log \tilde{n})$ for $x \in [1/\tilde{n}, 1]$. Let p_{half} denote the probability of drawing a value from $[z/2, z]$. For any $z \in [2/\tilde{n}, 1]$, $p_{\text{half}} = \int_{z/2}^{z} p_{\tilde{n}}(x)dx = 1/\log_2 \tilde{n}$, which is independent of z. Now, $1/\log_2 \tilde{n} > 1/(3\log_2 n)$. The significance of p_{half}: regardless of the current clockwise-distance to the destination, it is the probability that any single long-distance link will cut the distance by at least half. The number of links to consider before the current distance diminishes by at least half is at most $3\log_2 n$. With k links per node, the expected number of nodes to consider before the current distance is at least halved is at most $\lceil (3\log_2 n)/k \rceil$, which is less than $(6\log_2 n)/k$ for $k \leq \log_2 n$. Since the smallest arc-length is $\Theta(\frac{1}{n^2})$, the maximum number of times the original distance could possibly be halved is $\log_2(cn^2)$, for some constant c. Therefore, the expected length of route is at most $6(\log_2 n)(\log_2(cn^2))/k = O(\frac{1}{k}\log^2 n)$ hops. $\square$

7.5.4 Join and Leave Protocols

Join: To join the network, a new node must know at least one existing member. It then chooses its own id x from $[0, 1)$ uniformly at random. Using the Routing Protocol, it identifies node y, the current manager of x. It then runs the Estimation Protocol using $s = 3$, updating the estimates of three other nodes as well. Let $\tilde{n}_x$ denote the estimate of n thus produced. Node x then uses pdf $p_{\tilde{n}_x}$ to establish its long distance links. Since each link establishment requires a lookup that costs $O(\frac{1}{k}\log^2 n)$ messages, the total cost of k link establishments is $(\log^2 n)$ messages. The constant hidden behind the big-O notation is actually less than 1. See Section 7.6.6 for costs determined experimentally.

Leave: The departure of a node x is handled as follows. All outgoing and incoming links to its long-distance neighbors are snapped. Other nodes whose outgoing links to x were just broken, re-instate those links with other nodes. The immediate neighbors of x establish short links between themselves to maintain the ring. Also, the successor of x initiates the Estimation Protocol over $s = 3$ neighbors, each of whom also updates its own estimate of n. The departure of a node requires an average of k incoming links to be re-established. The expected cost is $O(\log^2 n)$ messages. Again, the constant hidden behind the big-O notation is less than 1. See Section 7.6.6 for costs determined experimentally.

7.5.5 Re-linking Protocol

Each node x in the network maintains two values: $\tilde{n}_x$, its current estimate of n and $\tilde{n}_x^{link}$, the estimate at which its long distance links were last established. Over its lifetime, $\tilde{n}_x$ gets updated due to the Estimation Protocol being initiated by other nodes. Whenever $\tilde{n}_x \neq \tilde{n}_x^{link}$, it is true that the current long distance links of x correspond to a stale estimate of n. One solution is to establish all links afresh. However, if a node were to re-link on *every* update of $\tilde{n}_x$, traffic for re-linking would be excessive. This is because re-establishment of all k long distance links requires $O(\log^2 n)$ messages.

Re-linking Criterion: A compromise re-linking criterion that works very well is to re-link only when the ratio $\tilde{n}_x/\tilde{n}_x^{link} \notin [\frac{1}{2}, 2]$. The advantage of this scheme is that as n steadily grows or shrinks, traffic for re-linking is smooth over the lifetime of the network. In particular, if nodes arrive sequentially and each node knows n precisely at all times, then the number of nodes re-linking at any time would be at most one. We experimentally show that even in the presence of imprecise knowledge of n, the re-linking cost is smooth over the lifetime of a network. However, we also show that the benefits of re-linking are marginal.

7.5.6 Greedy Routing with 1-Lookahead

Two nodes connected by a long link could periodically exchange some information piggy-backed on keep-alives. In particular, they could inform each other about the positions of their respective long distance neighbors on the circle. Thus a node can learn and maintain a list of all its neighbor's neighbors. We call this the lookahead list. The lookahead list helps to improve the choice of neighbor for routing queries. Let u denote the node in the list that takes a query closest to its final destination. Then the query is routed to that neighbor that contains u in its neighbor set. Note that we do not route directly to u. Upon receiving a forwarded lookup request, a neighbor makes a fresh choice for its own best neighbor to route to. We experimentally show that 1-Lookahead effectively reduces average latency by roughly 40%.

What is the cost of 1-Lookahead? The size of the lookahead list is $O(k^2)$. The number of long links remains unchanged because a node does not directly link to its neighbors'

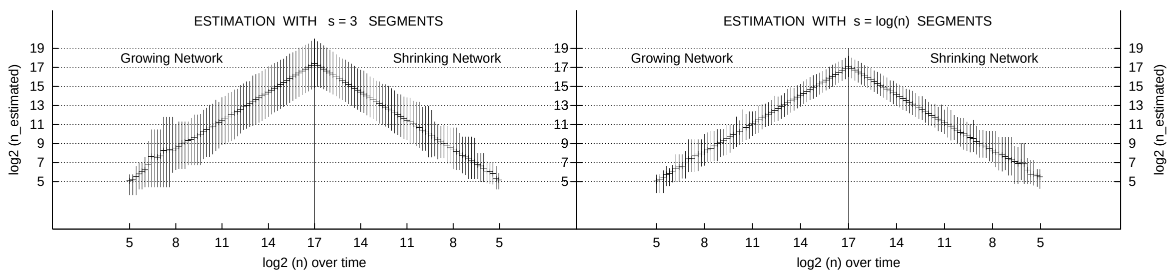


Figure 7.2: *Quality of estimated value of n as the network first expands and then shrinks. Each vertical line segment plots the average along with an interval that captures 99% of the distribution.*

neighbors; it just remembers their IDs. However, arrival and departure of any node requires an average of $k(2k+2)$ messages to update lookahead lists at $k(2k+2)$ nodes in the immediate neighborhood. These messages need not be sent immediately upon node arrival/departure. They are sent lazily, piggy-backed on normal routing packets or keep-alives exchanged between pairs of nodes. Lazy update of lookahead lists might introduce temporary inconsistencies. This is acceptable because routing does not crucially depend on these lists. Lookaheads just provide a better hint.

We could employ ℓ-Lookahead in general, for $\ell > 1$. However, the cost of even 2-Lookahead becomes significant since each update to a link would now require $O(k^3)$ additional messages for updating lookahead lists. If ℓ were as large as $O(\log^2 n)$, each node could effectively compute the shortest path to any destination.

7.6 Experiments

In this Section, we present results from our simulation of Symphony on networks ranging from 2^5 to 2^{15} nodes. We systematically show the interplay of various variables (n, k and s), justifying our choices. We study four kinds of networks: A Static network with n nodes is constructed by placing n nodes on a circle, splitting it evenly into n segments. Knowledge of n is global and accurate. An Expanding network is one that is constructed by adding nodes to the network sequentially. An estimate of n is used to establish long distance links. An Expanding-Relink network is simply an Expanding network in which nodes re-establish links using the re-linking criterion mentioned in Section 7.5.5. Finally, a Dynamic network is one in which nodes not only arrive but also depart. We describe the

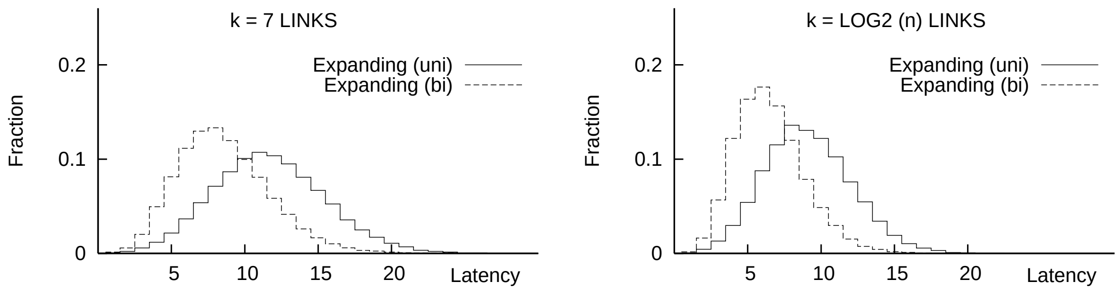


Figure 7.3: *Latency distributions for a network with* 2^{14} *nodes. "Uni" denotes Clockwise-Greedy Routing with Uni-directional Links. "Bi" denotes Absolute-Greedy Routing with Bi-directional Links.*

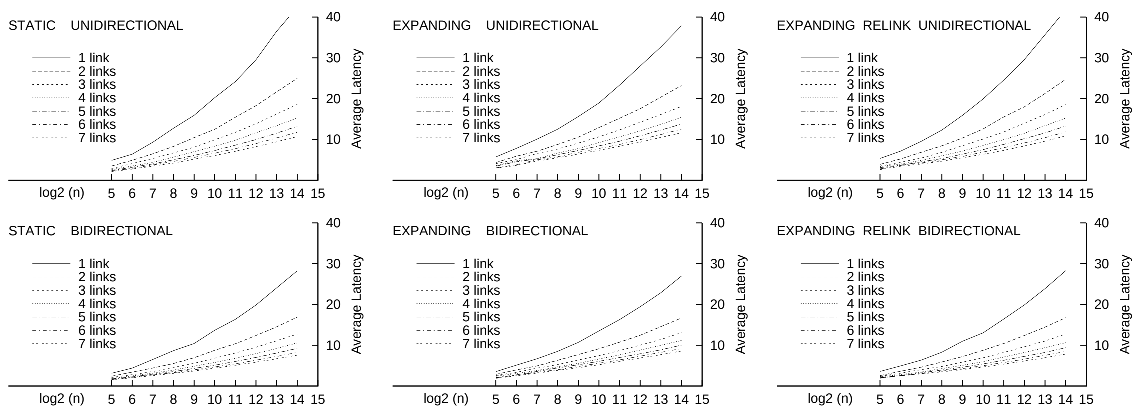


Figure 7.4: *Average latency for various numbers of long distance links and* n *ranging from* 2^5 *and* 2^{14}.

exact arrival and departure distributions in Section 7.6.4.

7.6.1 Estimation Protocol

Figure 7.2 shows performance of the Estimation Protocol when a network grew from zero to 2^{17} nodes and then shrank. Each vertical segment in the figure captures 99% of the nodes. The Estimation Protocol tracks n fairly well. The estimate is significantly improved if we use $s = \log \tilde{n}$ neighbors, where $\tilde{n}$ itself is obtained from any existing node. However, the impact on average latency is not significant, as we show in Section 7.6.3. All experiments described hereafter were conducted with $s = 3$.

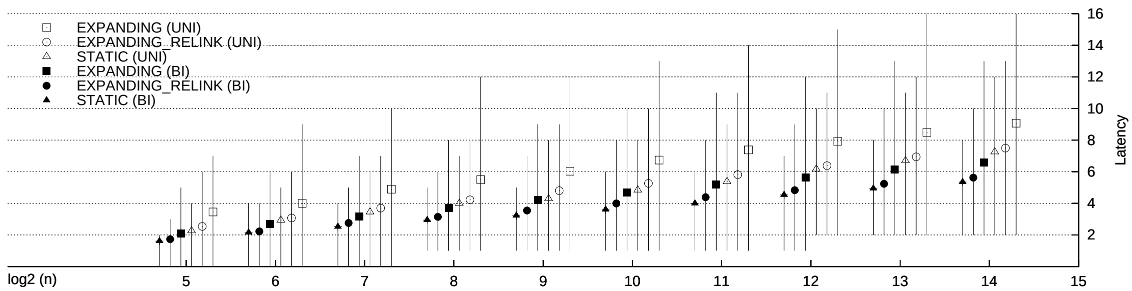


Figure 7.5: *Latency for various networks with* $\log_2 \tilde{n}$ *links per node. Each vertical segment plots the average along with an interval that captures* 99% *of the distribution. "Uni" denotes* Clockwise-Greedy Routing with Uni-directional Links. *"Bi" denotes* Absolute-Greedy Routing with Bi-directional Links.

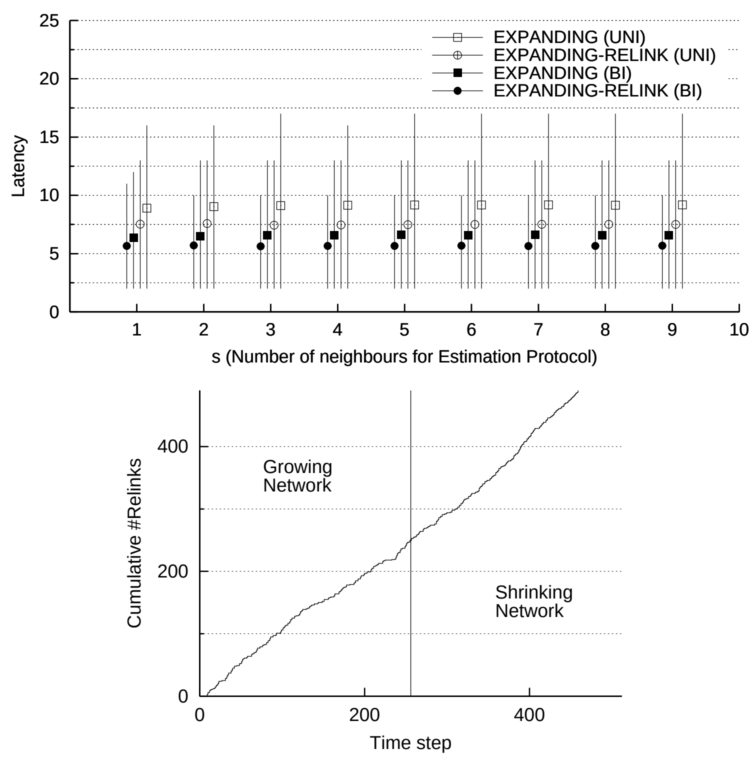


Figure 7.6: **Left:** *Latency for* EXPANDING *networks using Estimation Protocol with various values of s, the number of neighbors contacted for estimating n.* **Right**:*Cumulative number of re links for a network that first expands from* 0 *to* 256 *nodes and then shrinks back to* 0. *At every time step, exactly one node joins or leaves.*

7.6.2 Routing Protocol

Figure 7.4 on page 133 plots the average latency for three networks: STATIC, EXPANDING and EXPANDING-RELINK. The number of links per node is varied from 1 to 7. Increasing the number of links from 1 to 2 reduces latency significantly. However, *successive additions have diminishing returns*. This is one reason that re-linking has marginal benefits. However, Absolute-Greedy Routing with Bi-directional Links is a good idea as it improves latency by roughly 25% to 30%. Figure 7.3 on page 133 shows the latency distribution for various networks with either 7 or $\log_2 \tilde{n}$ links each. The variance of latency distribution is not high. Having $\log_2 \tilde{n}$ links per node not only diminishes average latency but also the variance significantly. Figure 7.5 on the preceding page plots latency for a network in which each node maintains $\log_2 \tilde{n}$ links. The vertical segments capture 99% of node-pairs. For a given type of network, average latency grows linearly with $\log n$, as expected.

7.6.3 Re-linking Protocol

In Figure 7.6 on the facing page, we plot average latency as s, the number of neighbors in the estimation protocol, varies. We observe that average latency is relatively insensitive to the value of s used. This justifies our choice of $s = 3$. Figure 7.6 on the preceding page also shows the cost of re-linking over the lifetime of a network that first expands to 256 nodes and then shrinks back to zero. Exactly one node arrives or leaves at any time step. We chose a network with small n for Figure 7.6 on the facing page to highlight the kinks in the curve. For large n, the graph looks like a straight line. The cost of re-linking is fairly smooth.

7.6.4 Dynamic Network

A DYNAMIC network is one in which nodes arrive and depart. We studied a network with n = 100K nodes having $\log \tilde{n}$ neighbors each. Each node alternates between two states: alive and asleep. Lifetime and sleep-time are drawn from two different exponential distributions with means 0.5 hours and 23.5 hours respectively. We grow the node pool linearly over a period of one day so that all 100K nodes are members of the pool at the end of the first day. During the second day, the pool remains constant. The third day sees the demise of random nodes at regular intervals so that the pool shrinks to zero by the end of 72 hours. The average number of nodes that participate in the ring at any time is

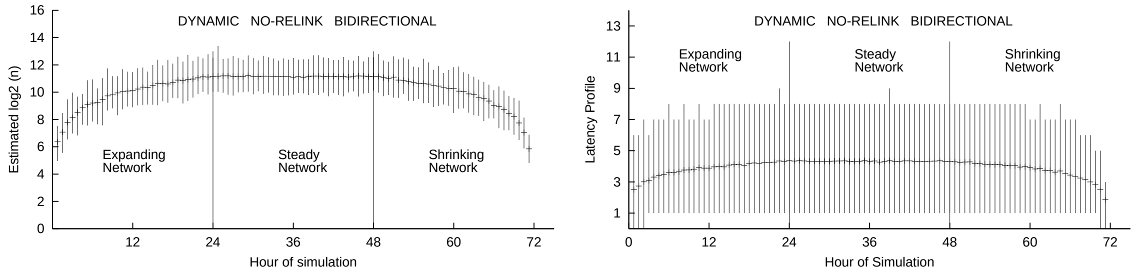


Figure 7.7: *Performance of a* DYNAMIC *network of 100K nodes with* $\log \tilde{n}$*-links using the Estimation Protocol but no re-linking. Each node is alive and asleep on average for 0.5 hours and 23.5 hours respectively. The node pool linearly increases to 100K over the first day. The pool is steady on the second day. A random node departs at regular intervals on the third day until the network shrinks to zero. Each vertical segment plots the average along with the range of values that covers* 99% *of the distribution.*

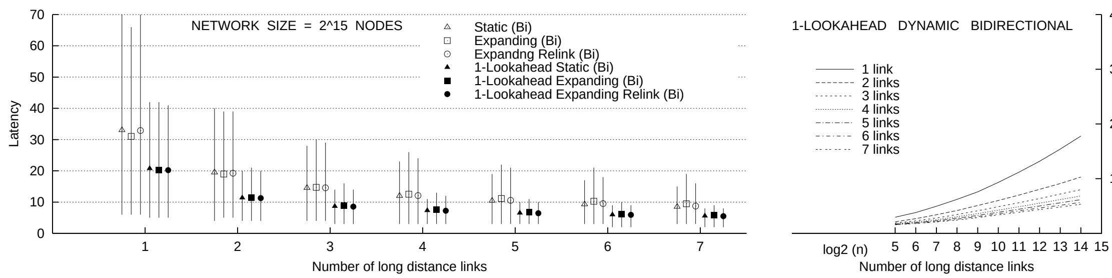


Figure 7.8: *Impact of using 1-Lookahead in routing in a typical network with* 2^{15} *nodes.*

$\frac{0.5}{0.5+23.5} \times 100K \approx 2K$. From Figure 7.7, we see that the Estimation Protocol is able to track n sufficiently accurately and that the average latency was always less than 5 hops.

We wish to point out that the network we simulated is very dynamic. The set of nodes at any point of time is quite different from the set of nodes one hour earlier. This is because the average lifetime of any node is only 0.5 hour. Hosts in real-life P2P network have average lifetimes of this order. However, a few hosts have much longer lifetimes (for measurements of Gnutella and Kazaa users, see Ripeanu *et al* [RFI02], Saroiu *et al* [SGG02] and Gummadi *et al* [GDS+03]). Our current model is simple but sufficient to highlight the stability of Symphony in the presence of high activity.

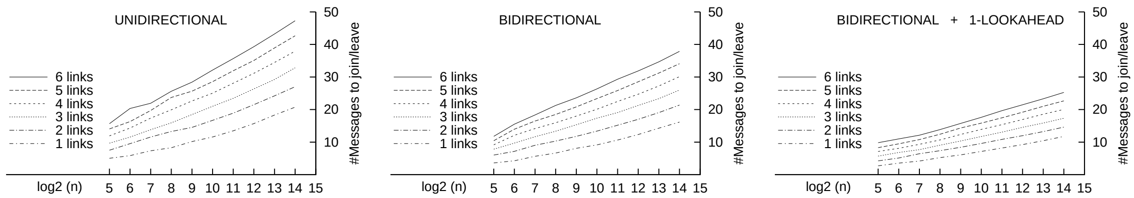


Figure 7.9: *Cost of joining and leaving in a network with* $n = 2^{15}$ *nodes.*

7.6.5 Lookahead

Figure 7.8 on the facing page shows the efficacy of employing 1-Lookahead when k is small. Average latency diminishes by around 40% with 1-Lookahead. Moreover, the spread of latency distribution (captured by vertical line segments in the graph) shrinks. For a network with 2^{15} nodes, average latency is 7.6 with $k = 4$. We also simulated 1-Lookahead with $k = \log \tilde{n}$ links per node and saw average latency drop to 4.4.

Note that 1-Lookahead does not entail an increase in the number of long links per node. Only neighbor-lists are exchanged between pairs of nodes periodically. This does not incur extra cost because increments to neighbor-lists are piggy-backed on normal routing traffic or keep-alives.

7.6.6 Cost of Joining and Leaving

Figure 7.9 plots the cost of joining and leaving the network. Whenever a node joins/leaves, k long distance links have to be created apart from updates to short links. Join/leave cost is proportional to the number of links to be established. The cost diminishes as average lookup latency drops. When using 1-Lookahead, there are an additional $k(2k+2)$ messages to update lookahead lists. However, these are exchanged lazily between pairs of nodes, piggy-backed on keep-alives. The cost of join/leave is $O(\log^2 n)$. Figure 7.9 clearly shows that the constant in the big-O notation is less than 1. For example, in a network of size 2^{14}, we need only 20 messages to establish $k = 4$ long links.

7.6.7 Load Balance

Figure 7.10 plots the number of messages processed per node in a network of size 2^{15} corresponding to 2^{15} lookups. Each lookup starts at a node chosen uniformly at random.

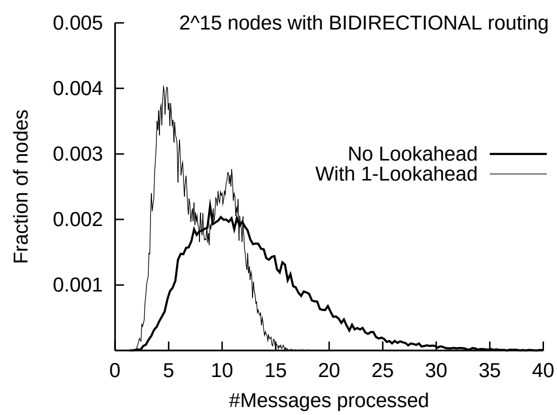


Figure 7.10: *Bandwidth profile in a network with* $n = 2^{15}$ *nodes with* $k = 4$ *links per node. Each node looks up one random hash key.*

The hash key being looked up is also drawn uniformly from $[0, 1]$. The routing load on various nodes is relatively well balanced. Both the average and the variance drop when we employ 1-Lookahead. Curiously, the distribution is bi-modal when 1-Lookahead is employed.

7.6.8 Resilience to Link Failures

Figure 7.11 explores the fault tolerance of our network. The top graph plots the fraction of queries answerable when a random subset of links (short as well as long) is deleted from the network. The bottom graph studies the impact of removing just long links. The slow increase in average latency is explained by Figure 7.4 which demonstrated diminishing returns of additional links. Figure 7.11 clearly shows that deletion of short links is much more detrimental to performance than deletion of long links. This is because removal of short links makes some nodes isolated. Removal of long links only makes some routes longer. Figure 7.11 suggests that for fault tolerance, we need to fortify only the short links that constitute the ring structure. We need not have backups for the long-distance links. In fact, this justifies the original design decision made for Dipsea (see Figure 1.1 for its overall architecture).

7.6.9 Comparison with k Random Links

Figure 7.12 compares average latency for Symphony with a network where nodes form outgoing links with other nodes uniformly at random. The figure clearly shows that the

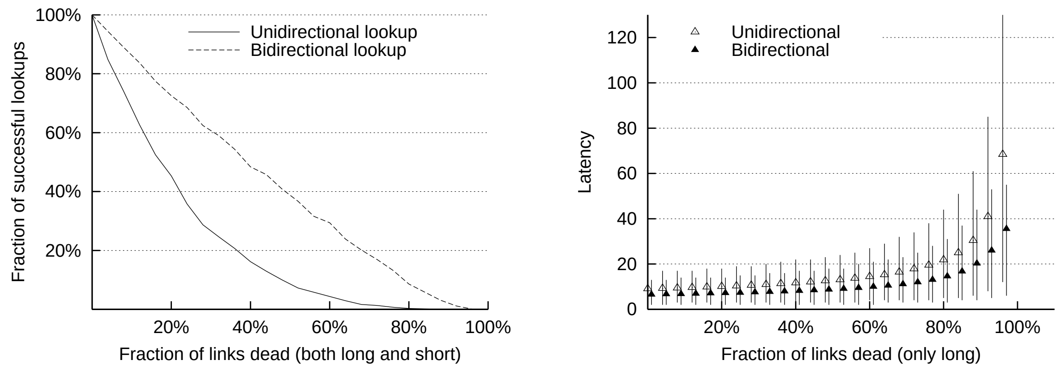


Figure 7.11: *Studying fault tolerance in a network of 16K nodes with* $\log \tilde{n}$ *long distance links per node. The top graph shows percentage of successful lookups when a fraction of links (short and long) are randomly deleted. The bottom graph shows increase in latency when only long links are randomly deleted.*

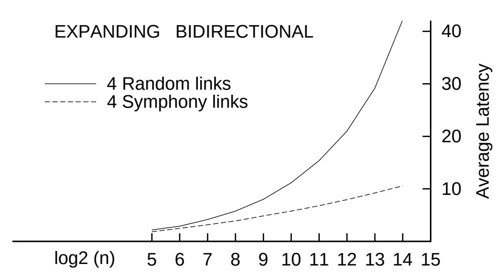


Figure 7.12: *Comparison of Symphony with a network where each node links to* k *other nodes chosen uniformly at random. Network size* $n = 2^{15}$ *nodes.*

TCP Conn.	Lookup Latency	Routing Network	TCP Conn.	Route Length	Notes
$2d$	$(d/2)n^{\frac{1}{d}}$	CAN	20	14.14	Fixed #dimensions
$2\log_2 n$	$(\log_2 n)/2$	Chord	30	7.50	Fixed #links
10	$\log_2 n$	Viceroy	10	15.00	Fixed #links
$\frac{2^b-1}{b}\log_2 n$	$(\log_2 n)/b$	Tapestry	56	3.75	with b=4 digits
$\frac{2^b-1}{b}\log_2 n$	$(\log_2 n)/b$	Pastry	22	7.50	with b=2 digits
			56	3.75	with b=4 digits
$2k+2$	$c(\log^2 n)/k$	Symphony	10	7.56	k=4, bidirectional with 1-lookahead
			56	3.75	k=27, bidirectional with 1-lookahead

Table 7.1: *Comparison of various routing networks over* 2^{15} *nodes. Latencies are measured in terms of hops.*

obvious idea of choosing k uniformly random long distance neighbors does not scale since the path length grows as $O(\sqrt{n/k})$.

7.7 Comparison and Analysis

Symphony is a simple randomized routing network that scales well and offers low lookup latency with only a handful of TCP connections per node. The cost of joining and leaving the network is small. We now highlight features unique to Symphony.

7.7.1 Low State Maintenance

Table 7.1 lists lookup latency vs. degree for various DHT routing networks. Low-degree networks are desirable for several reasons. First, fewer links in the network reduce the average number of open connections at servers and reduce ambient traffic corresponding to pings/keep-alives and control information. Second, arrivals and departures engender changes in DHT topology. Such changes are concomitant with the state update of a set of nodes whose size is typically proportional to the average degree of the network. Fewer links per node translates to smaller sets of nodes that hold locks and participate in some coordination protocol for distributed state update. Third, small out-degree translates to smaller boot-strapping time when a node joins and smaller recovery time when a node leaves without notice. Finally, it should be easier to isolate faults in low degree networks, making debugging faster.

7.7.2 Fault Tolerance

Symphony does not create any redundant long distance links for fault tolerance. There are no *backup* long links. It is only the short links that are fortified by maintaining connections with f successors per node, where f is a design parameter. The long links contribute to the *efficiency* of the network; they are not critical for correctness (see Section 7.6.8). Protocols like Pastry, Tapestry and CAN maintain two to four backup links for *every* link a node has. A glance at Table 7.1 reveals that the overhead of redundant links for fault tolerance is significantly less for Symphony than other protocols. Having fewer links per node has several other benefits that we described in the preceding Section.

7.7.3 Smooth Tradeoff between Degree and Latency

Symphony provides a *smooth tradeoff* between the number of links per node and average lookup latency. It appears to be the only protocol that provides this tuning knob *even* at run-time. Symphony does not dictate that the number of links be identical for all nodes. Neither is the number stipulated to be a function of current network size nor is it fixed at the outset, unlike CAN [RFHK01], Chord [SMK+01], Pastry [RD01a] or Tapestry [ZHS+04]. We believe that these features of Symphony provides three benefits:

Support for Heterogeneous Nodes: Each node is merely required to have a bare minimum of two short-distance links. The number of long-distance links can be chosen for each individual node according to its available bandwidth, average lifetime, or processing capability. All the other DHT protocols specify the exact number and identity of neighbors for each node in the network. It is not clear how they would accommodate nodes with variable degrees. Symphony's randomized construction makes it adapt naturally to heterogeneous nodes.

Incremental Scalability: Symphony scales gracefully with network size. The Estimation Protocol provides each participant with a reasonably accurate estimate of network size. It is possible for nodes to adapt the number of long distance links in response to changes in network size to guarantee small average lookup latency. This obviates the need to estimate in advance the maximum size of the network over its lifetime.

Flexibility: An application designer who uses a distributed hash table (DHT) would want to make its implementation more efficient by leveraging knowledge unique to the problem scenario. For example, the specifics of the network topology at hand, or the behavior of participating hosts, or a priori knowledge about the load on the DHT might be known. If the DHT itself has a rigid structure, the application designer is severely constrained. Symphony allows the number of links to be variable. All outgoing links are *identical* in the sense that they are drawn from the same probability distribution function. We believe that the randomized nature of Symphony poses few constraints as compared with other protocols.

7.7.4 Comparison with Other Protocols

We compare Symphony with other DHT routing networks over $n = 2^{15}$ nodes.

(a) CAN [RFHK01] can route among n nodes with an average latency of $(d/2)n^{\frac{1}{d}}$. The optimal value of d for $n = 2^{15}$ nodes is 10 resulting in an average latency of 14.14. The average number of TCP connections is $2d = 20$. Dimensionality in CAN is fixed at the outset. It is not clear how dimensionality can be dynamically changed as the network expands or shrinks. Thus, CAN nodes would have 20 TCP connections each even if the network size is small.

(b) Chord [SMK+01] stipulates that every node in a network with 2^{15} must have $\log_2 n = 15$ outgoing links each, with the result that average latency is 7.5. In terms of TCP connections, nodes have $2\log_2 n = 30$ connections each. Among existing DHT protocols, Symphony is closest in spirit to Chord. Chord could borrow ideas from Symphony for better performance. For example, Chord currently uses clockwise routing using unidirectional links. It can be modified to employ Symphony-style greedy routing over bidirectional links that minimizes absolute distance to the target at each hop.

(c) Pastry [RD01a], with a digit size of 2 bits, would need an average of 22 TCP connections per node for average latency 7.5. Pastry can improve the latency to 3.75, but only with as many as 56 TCP connections per node. The digit size is a parameter that is fixed at the outset. For fault tolerance, Pastry maintains backup links for every link in its routing table.

(d) Tapestry [ZHS+04] uses 4-bit digits resulting in average lookup latency of 3.75 with 56 links per node. Tapestry is very similar to Pastry. The digit size is a parameter that is fixed at the outset. For fault tolerance, Pastry maintains backup links for every link in its routing table.

(e) Viceroy [MNR02] maintains seven links per node, irrespective of n, the size of the network. Each node has two neighbors along two rings, one up-link and two down-links.

Four of these links are bidirectional, three are unidirectional. Thus, a Viceroy node would actually have an average of $t = 10$ TCP connections per node. For $n = 2^{15}$, the average latency in Viceroy would be at least $\log(n) = 15$. This corresponds to an average 7.5 levels to reach *up* to the highest ring and another 7.5 levels to come *down* to the ring at the right level. Viceroy and Mariposa (see Chapter 9) are much more complex than Symphony.

(f) Symphony offers a wide variety of choices for the number of TCP connections for a fixed value of $n = 2^{15}$ nodes. Figure 7.8 shows that the average latency with $k = 4$ long links with 1-Lookahead and bidirectional routing is 7.6. Such a topology results in 10 TCP connections per node on average. As k increases, Symphony's average latency reduces. Symphony does not use backup links for long distance links.

7.7.5 The Role of Lookahead

Lookahead is of little value to deterministic routing networks like tori (used in CAN), hypercubes (used in Pastry and Tapestry) or Chord. This is because the structure of the graph is global information and routes follow shortest paths. However, in a randomized topology like Symphony, 1-Lookahead is valuable, as exemplified by our experiments. In a randomized network, each node makes links based on random-bits generated locally – these random bits are not global information. In Chapter 8, we investigate formally the role of 1-Lookahead in Symphony and randomized variants of Chord and the hypercube. We will establish that with $O(\log n)$ links per node, the average length of routes *without* 1-Lookahead is $\Omega(\log n)$ hops. However, with 1-Lookahead, the average route length drops to $O(\log n / \log\log n)$, which is asymptotically optimal.

7.8 Summary and Future Directions

We presented Symphony, a simple randomized routing network for DHTs. With k outgoing links per node, greedy routing in Symphony results in routes of length $O(\frac{1}{k}\log^2 n)$ on average. Through a series of systematic experiments, we have shown that Symphony scales well, has low lookup latency and maintenance cost with only a few neighbors per node. In particular, $s = 3$ neighbors suffice for the Estimation Protocol and $k = 4$ long-distance links with Absolute-Greedy Routing and 1-Lookahead are sufficient for low latencies in networks as big as 2^{15} nodes. We formally analyze 1-Lookahead in Chapter 8.

Future directions: It would be interesting to formally establish other properties of Symphony: What is its bisection width? What is the edge-congestion caused by greedy routing with/without 1-lookahead? What are the relationships between Symphony and well-known deterministic routing networks? Symphony has non-uniform in-degrees of nodes. Can this be fixed in a practical system? Which probability distribution function results in the least average path length with greedy routing?

Chapter 8

Greedy Routing with Lookahead

In this Chapter, we study randomized routing networks. These networks can be used in the module named "Choice of Long-distance Links" of Dipsea (see Figure 1.1 on page 2 for a block-diagram of its architecture). All of the networks that we discuss are node-symmetric, quite easy to describe and use simple but efficient routing strategies. In terms of the tradeoff between the number of out-going links per node and the average length of routes, the randomized routing networks described in this Chapter are competitive with the best-known deterministic routing networks.

8.1 Introduction

Randomized routing networks arise in two different contexts: as graph models to explain the small-world phenomenon, and as routing networks for peer-to-peer systems. We briefly outline recent developments in these two areas.

Small World Phenomenon

A widely-held belief pertaining to social networks is that any two people in the world are connected via a chain of six acquaintances†. A quantitative study of the phenomenon was started in 1960s by Milgram [M67] who asked people to send letters to unfamiliar targets only through social acquaintances. Milgram's experiments, and later work by Pool and Kochen [PK78] confirmed that random pairs of individuals are

†According to Barabási [B02], the idea of *six-degrees of separation* may have its origins in a short story "Chains" by the Hungarian writer Frigyes Karinthy from 1929; this idea has since been retold and recast many times in literature, popular press as well as scientific studies.

indeed connected by short chains of acquaintances. In fact, Milgram's experiments also demonstrated that individuals are able to *route* messages to unknown targets. Such routing properties were also observed by Dodds *et al* [DMW03] who asked people to forward e-mails to their acquaintances so that the e-mail eventually reaches a target unknown to the source.

To model the routing aspects of the small-world phenomenon, Kleinberg [K00] constructed a family of random graphs. The graphs not only have small diameter (to model the "six degrees of separation") but also allow short routes to be discovered on the basis of local information alone (to model Milgram's observation that "messages can be routed to unknown individuals efficiently"). Specifically, Kleinberg considered a 2D $n \times n$ grid with n^2 nodes. Each node is equipped with a small set of "local" contacts and one "long-range" contact drawn from a harmonic distribution. With greedy routing, the path-length between any pair of nodes is $O(\log^2 n)$ hops, with high probability[†]. Local knowledge available to a node suffices for greedy routing since a message is forwarded along that out-going link which takes it *closest* to the destination.

Randomized Peer-to-Peer Networks

Symphony [MBR03] is a successful adaptation of Kleinberg's construction [K00] to arrive at a randomized P2P routing network. The idea is to place nodes in a ring (instead of a 2D grid) and to equip each node with *multiple* "long-distance" links (instead of just one). An adaptation of Kleinberg's construction is apt for P2P routing because individual nodes have *low degree* and yet are able to route messages efficiently using only *local information.*

An important distinction between deterministic and randomized networks pertains to information available to different nodes about the global network. In deterministic networks, each node has knowledge of the entire network. However, in the randomized networks we study in this Chapter, each node possesses knowledge of only its own set of long-distance links. Therefore, there is interest in devising efficient *decentralized routing protocols* for randomized routing networks that use only local information. Clockwise greedy routing is one such protocol, where the idea is to forward a message

[†]By "with high probability" (w.h.p.), we mean "with probability at least $1 - O(n^{-\lambda})$ for an arbitrary constant $\lambda > 1$".

along that out-going link that minimizes the remaining clockwise distance to the destination.

Recently, three more randomized routing networks have been devised: Randomized-Hypercube, Randomized-Chord and skip-graphs. All of these have $\Theta(\log n)$ out-going links per node, and use some form of greedy routing, which is known to take $O(\log n)$ hops on average. Randomized-Hypercube [GGG+03, CDHR03], as the name suggests, is a randomized variant of hypercubes. Randomized-Chord [GGG+03, ZGG03] is a variation on a recently-devised deterministic graph topology called Chord [SMK+01, GM04]. Skip-graphs [AS03], also known as SkipNet [HJS+03], build upon the intuition inherent in skip-lists [P90].

Remark: Viceroy [MNR02] and Mariposa (see reference [M03] or Chapter 9) are two randomized networks that we do not discuss in this Chapter. The design philosophy underlying these two networks is quite different — see the beginning of Chapter 9 for details.

Characterization of the tradeoff between the number of links per node and the average number of routing hops is of great interest to designers of DHT routing networks [RSS02]. The Degree-Diameter Problem, studied in extremal graph theory, seeks to identify the largest graph with diameter Δ, with each node having out-degree at most d (see [D04] for a survey). A well-known upper bound for the problem is $1 + d + d^2 + \cdots + d^\Delta = \frac{d^{\Delta+1}-1}{d-1}$, also known as the Moore bound. A general lower bound is $d^\Delta + d^{\Delta-1}$, achieved by Kautz digraphs [K68,K69], which are slightly superior to de Bruijn graphs [dB46] whose size is only d^Δ. Two consequences of these results are: (a) with out-degree d per node, the diameter of any graph on n nodes is $\Omega(\log n/\log d)$, and (b) there exist constructions (high-degree butterfly networks and de Bruijn graphs, for example) whose diameter is $O(\log n/\log d)$. These graphs have been studied extensively in the context of routing in parallel machine architectures (see the book by Leighton [L92]). Our focus in this Chapter is on *randomized* routing networks. The $\Omega(\log n/\log d)$ bound applies to these networks as well. This means that when $d = \Theta(\log n)$, it *might* be possible to route in $\Theta(\log n/\log\log n)$ hops on average.

Our Contributions

In this Chapter, we address two questions pertaining to various randomized routing networks (Symphony, Randomized-Hypercubes, Randomized-Chord and skip-graphs):

a) *Is greedy routing optimal?*

We show that greedy routing, which is known to take $O(\log n)$ hops on average, is asymptotically sub-optimal. We do so by furnishing a matching lower bound of $\Omega(\log n)$ for Randomized-Hypercube and Randomized-Chord. For Symphony with k links per node, we show that greedy routing requires $\Omega(\frac{1}{k}\log^2 n)$ hops on average. This matches the upper-bound of $O(\frac{1}{k}\log^2 n)$ proved earlier in Chapter 7. For skip-graphs, a lower bound of $\Omega(\log n)$ is shown in reference [MNW04].

b) *What is the role of lookahead upon greedy routing?*

"Greedy with lookahead" was first proposed as a heuristic in Symphony [MBR03]. The idea is to allow a node to gain knowledge of its neighbor's neighbors for making better routing decisions. Such knowledge reduces the average route lengths significantly (see Chapter 7 for graphs). These observations motivated a theoretical investigation into whether look-ahead reduces the asymptotic routing complexity, or just diminishes the average by some constant-factor. Greedy routing, as we noted above, takes $\Theta(\log n)$ hops on average. Does look-ahead reduce the average to $O(\log n/\log\log n)$?

We show that greedy with lookahead routing, requires $O(\log n/\log\log n)$ hops in Randomized-Hypercubes and Randomized-Chord. For Symphony with k links per node, average route length is only $O((\log^2 n)/(k\log k))$. Similar results for skip-graphs are proved in reference [MNW04].

Additional results proved in this Chapter are the following:

a) Using the Bit-Collection protocol (see reference [M03] or Section 8.5 for more details), $R(n) = O(\log n)$ hops for both Sparse-Chord and Sparse-Hypercube (which we define in §8.5).

b) With k links per node, $R(n) = O(\log^2 n/(k\log k))$ hops for GREEDY with 1-LOOKAHEAD routing in Symphony/Symphony*.

Summary of Results

In §8.2, we formally define GREEDY routing and various randomized routing networks.

In §8.3, we analyze GREEDY routing in various randomized networks.

In §8.4, we analyze GREEDY with 1-LOOKAHEAD routing.

In §8.5, we study the "Bit Collection protocol" for sparse-Chord and sparse-hypercube.
In §8.6, we discuss related work.
In §8.7, we discuss system issues pertaining to the randomized networks we analyze.
In §8.8, we summarize our results and present directions for future work.

8.2 Definitions

We will study directed graphs over n nodes placed in a circle, labeled 0 through $n-1$. We will also assume that n is a power of two. In a peer-to-peer (P2P) system, the set of nodes is dynamic and not necessarily a power of two. However, as outlined in the architecture of Dipsea (see Figure 1.1 on page 2 for a block-diagram), the Emulation Engine (described in Chapter 4) absorbs the design complexity associated with scale and dynamism, allowing us to focus on families of routing networks defined over power-of-two nodes.

Distance Functions

We will study graphs consisting of n nodes placed in a circle. Two *natural* distance metrics on such graphs are the clockwise-distance and the absolute-distance between pairs of nodes. The Hamming distance between node pairs is also a popular distance metric.

DEFINITION (Distance Function δ).

$$\delta_{clockwise}(u,v) = \begin{cases} v-u & v \geq u \\ n+v-u & otherwise \end{cases}$$

$$\delta_{absolute}(u,v) = \begin{cases} \min\{v-u, n+u-v\} & v \geq u \\ \min\{u-v, n+v-u\} & otherwise \end{cases}$$

$$\delta_{xor}(u,v) = |u \oplus v|$$

Deterministic Routing Networks

Both of the networks defined below are directed graphs with $n = 2^\ell$ nodes labeled 0 through $n-1$, arranged in a circle.

⋆ Hypercube

An edge from x to y exists iff the labels of x and y differ at exactly one bit-position.

- ⋆ Chord [SMK+01]

  An edge from x to y exists iff the clockwise distance from x to y is some power of two.

Randomized Routing Networks

All networks defined below are directed graphs with $n = 2^\ell$ nodes labeled 0 through $n-1$, arranged in a circle. Each node is connected to its successor by a *short-distance* link. The rest of the links are said to be *long-distance* and involve random choices.

- ⋆ Randomized-Hypercube [CDHR03, GGG+03]

  The out-degree of each node is ℓ. For each $1 \le i \le \ell$, node $\mathbf{x}$ makes a connection with node $\mathbf{y}$ defined as follows: The top $i-1$ bits of $\mathbf{y}$ are identical to those of $\mathbf{x}$. The i^{th} bit is flipped. Each of the remaining $\ell - i$ bits is chosen uniformly at random. The distance-function for routing is δ_{xor}.

- ⋆ Randomized-Chord [ZGG03, GGG+03]

  Node $\mathbf{x}$ makes ℓ connections as follows: Let $r(i)$ denote an integer chosen uniformly at random from the interval $[0, 2^i)$. Then for each $0 \le i < \ell$, node $\mathbf{x}$ creates an edge with node $(\mathbf{x} + 2^i + r(i)) \bmod n$. Each node has out-degree ℓ. The distance-function for routing is $\delta_{clockwise}$.

- ⋆ Symphony [MBR03]

  Node $\mathbf{x}$ establishes $k \ge 1$ *long-distance* edges as follows: For each edge, node $\mathbf{x}$ first draws a random number r from the probability distribution $p(x) = 1/(x \ln n)$ where $x \in [1, n]$ and then establishes a link with node $\lceil \mathbf{x} + r \rceil \bmod n$. The resulting graph is thus a multi-graph since two $\mathbf{x}$ could be connected to $\mathbf{y}$ by more than one edge. The distance-function for routing is $\delta_{clockwise}$.

- ⋆ Symphony* [MNW04]

  Node $\mathbf{x}$ establishes a *short-distance* edge with node $(\mathbf{x}+1) \bmod n$. Let δ denote a real number satisfying $\ln \delta = (\ln n)/k$. Let $\mathcal{I}_1 = [1, \delta]$. For $1 < i \le k$, let $\mathcal{I}_i = (\delta^{i-1}, \delta^i]$. For interval $\mathcal{I}_i$, let ϕ^i denote a probability distribution over integers in $\mathcal{I}_i$ such that the probability at integer d is proportional to $1/d$. For each $1 \le i \le k$, an edge is established with a node lying clockwise distance d away, where d is an integer drawn from ϕ^i. Edges are directed. The out-degree of each node is k.

Symphony* with $k = 1$ is identical to Kleinberg's construction [K00] in one dimension. For larger k, it is akin to chopping the probability distribution into k equal pieces and carrying out *stratified sampling*. Simulations indicate that Symphony* is slightly superior to Symphony in terms of average route lengths.

Greedy Routing with/without 1-Lookahead

There is an important distinction between deterministic and randomized routing networks. In deterministic networks, we assume that the structure of the network is global information. Therefore, messages can be sent along shortest paths. In a randomized routing network, each node makes links as a function of some random bits. We assume that these random bits are not global information. Therefore, it is not possible to send messages along shortest paths, in general. This motivates the need for *decentralized routing strategies* which allow a node to forward messages on the basis of as little knowledge of other nodes' random bits as possible. GREEDY routing is a natural decentralized routing strategy: a node forwards a message along that out-going edge that minimizes the *distance* remaining to the destination:

DEFINITION (Greedy Routing). *In graph (V, E) with distance function $\delta : V \times V \to \mathcal{R}^+$,* GREEDY *routing entails the following decision: Given a target node t, a node u with neighbors $N(u)$ forwards a message to its neighbor $v \in N(u)$ such that $\delta(v, t) = \min_{x \in N(u)} \delta(x, t)$.*

A variant of GREEDY routing is GREEDY with 1-LOOKAHEAD. The idea is to take a neighbors' neighbors into account to make better routing decisions.

DEFINITION (Greedy with 1-Lookahead Routing). *In graph (V, E) with distance function $\delta : V \times V \to \mathcal{R}^+$,* GREEDY *with 1-*LOOKAHEAD *routing entails the following decision: A node takes its neighbor's neighbors also into account when making routing decisions. Let $N(x)$ denote the neighbors of node x. Given target node t, node u first identifies node z such that $\delta(z, t) = \min_{x \in N(u)}\{\delta(x, t), \min_{y \in N(x)} \delta(y, t)\}$. If edge (u, z) exists, then node u forwards the message to node z. Otherwise, node v exists such that both (u, v) and (v, z) exist; u forwards the message to v, which then forwards the message to z.*

Notes: We use the prefix "1-" in 1-LOOKAHEAD to distinguish it from L-LOOKAHEAD in general, where long-range contacts of all nodes reachable within L hops are taken into account for making routing decisions. GREEDY with 1-LOOKAHEAD is slightly different from the lookahead-based routing scheme proposed in the Symphony paper [MBR03]. The difference is as follows: In GREEDY WITH 1-LOOKAHEAD, node **x** forwards the message to **u** *and* **u** forwards the message to **z**. However, in the scheme proposed in [MBR03], node **x** forwards the message to **u**, which then re-evaluates the best-possible neighbor based on lookahead. As a result, **u** may or may not forward the message to **z**.

Average Route Length $R(n)$

Let $r(\mathbf{x}, \mathbf{y})$ denotes the length of the route from node **x** to node **y**.
For deterministic topologies like Chord and hypercube,

$$R(n) \equiv n^{-2} \sum_{\mathbf{x},\mathbf{y} \in [0,n-1]} r(\mathbf{x}, \mathbf{y})$$

For randomized topologies, where $r(\mathbf{x}, \mathbf{y})$ is a random variable,

$$R(n) \equiv n^{-2} \sum_{\mathbf{x},\mathbf{y} \in [0,n-1]} \mathbf{E} r(\mathbf{x}, \mathbf{y})$$

Summary of Results

The following picture emerges in this Chapter:

A) *Deterministic topologies – the hypercube and Chord – have diameter* $\Theta(\log n)$.

In fact, GREEDY routing with distance function δ_{xor} is optimal for the hypercube, and GREEDY routing with distance function $\delta_{absolute}$ is optimal for Chord. Both route along shortest paths, with $R(n) = \Theta(\log n)$. 1-LOOKAHEAD offers no improvement.

B) *Randomization reduces the diameter to* $\Theta(\log n / \log \log n)$ *in expectation.*

Each of the following networks has diameter $\Theta(\log n / \log \log n)$: Randomized-Chord, Randomized-Hypercube, and Symphony/Symphony* with $k = \Theta(\log n)$ links per node. In comparison, the deterministic topologies (Hypercube and Chord) have diameter $\Theta(\log n)$.

A small diameter does not necessarily mean that there exist efficient decentralized routing algorithms. The following results study GREEDY with/without 1-LOOKAHEAD:

C) GREEDY *routing is unable to discover optimal routes in randomized networks.*

GREEDY routing, with the appropriate distance function, requires $R(n) = \Theta(\log n)$ hops on average for each of the following randomized networks: Randomized-Chord, Randomized-Hypercube, and Symphony/Symphony* with $k = \Theta(\log n)$ links per node.

D) GREEDY *with 1-*LOOKAHEAD *is asymptotically optimal for the randomized networks.*

Each of the following randomized networks requires $R(n) = \Theta(\log n / \log \log n)$ hops on average with GREEDY with 1-LOOKAHEAD routing: Randomized-Chord, Randomized-Hypercube, and Symphony/Symphony* with $k = \Theta(\log n)$ links per node.

E) *Empirically, the average route length of* GREEDY *with 1-*LOOKAHEAD *in Randomized-Chord, Randomized-Hypercube and Symphony is within 10% of average route lengths in de Bruijn networks with as many links per node.*

Results B), C) and D) above hold for three more randomized networks: SkipNet [HJS$^+$03], skip-graphs [AS03], and the Small-World Percolation Network (see reference [MNW04] or Section 8.5 for its definition).

8.3 Analysis of GREEDY Routing

In deterministic networks – hypercube and Chord – $R(n) = \Theta(\log n)$ for GREEDY routing. In a hypercube, $R(n) = \Theta(\log n)$ hops because routing from $\mathbf{x}$ to $\mathbf{y}$ amounts to setting the 1-bits of $\mathbf{x} \oplus \mathbf{y}$ to 0 in succession. In Chord, GREEDY routing from $\mathbf{x}$ to $\mathbf{y}$ amounts to setting the 1-bits in the binary representation of $(\mathbf{y} - \mathbf{x} + n) \bmod n$ to 0 in succession. Chord with each edge treated as undirected was analyzed in reference [GM04]. GREEDY routing is optimal (see Section 5.5.2 in Chapter 5) For both topologies, $R(n) = \ell/2 = \Theta(\log n)$. We now analyze GREEDY routing in randomized routing networks.

Theorem 8.1. $R(n) = \Theta(\log n)$ *for* GREEDY *routing in randomized hypercube.*

Proof. Consider the route from node $\mathbf{x}$ to node $\mathbf{y}$. Successive hops correspond to *fixing* the top bit of $\mathbf{z} \oplus \mathbf{y}$, where $\mathbf{z}$ is the current node. The probability that a specific bit requires fixing is half. It follows that the expected number of hops is $\ell/2 = \Theta(\log n)$. $\square$

For randomized Chord, we first prove a lemma related to the the following process: A particle starts at position m where $m > 0$ is an integer. At successive time-steps, the particle moves to a new position. When the current position is p, then the new position is a random variable X^p that ranges over the integers $0, \ldots, p$. We assume that for all $i \in [0, p-1]$, $Pr[X^p = i] > 0$. The process terminates when the particle reaches position 0. Let $T(m)$ denote the number of steps required for the process to terminate.

Lemma 8.3.1. *If for all $p \geq 2$, $Pr[X^p = i] \geq 2^{i-1} Pr[X^p = 0]$ for $i \in [0, p-2]$, then $\mathbf{E}T(m) = \Omega(m)$.*

Proof. Consider the same process but with each probability distribution X^p replaced by Y^p where $Pr[Y^p = i] = 2^{i-1} Pr[Y^p = 0]$ for $1 \leq i \leq p-2$ and $Pr[Y^p = p-1] = Pr[Y^p = p] = 0$. If $U(m)$ represents the number of steps required for the new process to terminate, then $\mathbf{E}T(m) \geq \mathbf{E}U(m)$. For each p, $Pr[Y^p = 0] = 1/2^{p-2}$ and $Pr[Y^p = i] = 2^{i-1}/2^{p-2}$ for $1 \leq i \leq p-2$. Using induction, it can be shown that for $m > 0$, $\mathbf{E}U(m) \geq cm$ for some constant $c < 1$:

Base case: $\mathbf{E}U(1) = 1$. Induction step:

$$
\begin{aligned}
\mathbf{E}U(m) &= 1 + \sum_{0 \leq i \leq m} \mathbf{E}U(i) Pr[X(m) = i] \\
&\geq 1 + \sum_{1 \leq i \leq m} ci Pr[X(m) = i] \\
&= 1 + \sum_{1 \leq i \leq m-2} ci 2^{i-m} \\
&\geq cm
\end{aligned}
$$

for a suitable constant c. ∎

Theorem 8.2. *$R(n) = \Theta(\log n)$ for* GREEDY *routing in randomized Chord.*

Proof. For a GREEDY route in randomized Chord, we define *phases* as follows: Phase 0 consists of one integer, namely 0. For $p \geq 1$, phase p consists of all integers in the interval $[2^{p-1}, 2^p - 1)$. Consider a message in phase $p \geq 2$, i.e., its remaining distance is d such that d belongs to phase p. For $0 \leq p' \leq p-2$, let $\phi(p \to p')$ denote the probability that the next phase is p'. By the definition of R-Chord, only two links at the current node decide the next hop for forwarding the message. For $d' \in [0, d - 2^p]$, the probability that the remaining distance is d' is exactly $1/2^{p-1}$. For $d' \in [d - 2^p + 1, d - 2^p + 2^{p-1}]$, the probability is exactly $(2^p - d - 1)/(2^{p-1} 2^{p-2})$. The latter probability is larger iff $d \leq 3 \cdot 2^{p-2} - 1$. In any case, $\phi(p \to p') \geq 2^{p'-1} \phi(p \to 0)$ for $0 \leq p' \leq p-2$. In fact, the equality holds if $d > 3 \cdot 2^{p-2}$.

There are $\log_2 d$ different phases if the initial distance is d. By applying Lemma 8.3.1, we deduce that the expected number of routing steps for distance d is $\Omega(\log d)$. Averaged over all possible values of d, we get that the average length of GREEDY routes is $\Omega(\log n)$. $\square$

Theorem 8.3. *$R(n) = \Theta(\log n)$ for* GREEDY *routing in Symphony.*

Theorem 8.4. *$R(n) = \Theta(\log n)$ for* GREEDY *routing in Symphony*.*

8.4 Analysis of GREEDY WITH 1-LOOKAHEAD Routing

For the deterministic topologies – Chord and Hypercube – $R(n) = \Theta(\log n)$ hops since GREEDY WITH 1-LOOKAHEAD is no better/worse than GREEDY. We now analyze randomized routing networks.

Theorem 8.5. *$R(n) = \Theta(\log n / \log \log n)$ hops for* GREEDY WITH 1-LOOKAHEAD *routing in randomized hypercube.*

Proof. Upper Bound: GREEDY WITH 1-LOOKAHEAD considers all nodes reachable in two hops and forwards the message (in at most two hops) to the node with the longest matching prefix with the destination. At least one bit gets fixed per hop. For $n = 2^\ell$ nodes, with $\ell > 2$, let $\ell = \ell_1 + \ell_2$, where $\ell_2 = \lceil \ell / \log_2 \ell \rceil$. The last ℓ_2 bits require no more than $O(\ell / \log \ell)$ hops even if they get fixed sequentially, one bit per hop. We will now show that fixing the top ℓ_1 bits require at most $2\ell_1 / \lfloor \log \ell_2 \rfloor$ hops in expectation.

Let $b < \ell_1$ denote the number of top bits that the current node shares with the destination $\mathbf{y}$. Thus, the current node differs from $\mathbf{y}$ in bit-position $b + 1$. Consider the neighbors of node $\mathbf{x}$ corresponding to the last $\ell - b - 1$ bits. Each of these neighbors has the top b bits in common with $\mathbf{x}$. Further, each of these neighbors establishes a link with some node which shares the top b bits, flips the next bit and chooses the trailing bits uniformly at random. Since $b < \ell_1$, there are at least ℓ_2 such neighbors. Therefore, the expected number of bits that get fixed is at least $\lfloor \log_2 \ell_2 \rfloor$.

The top ℓ_1 bits require no more than $2\ell_1 / \lfloor \log_2 \ell_2 \rfloor$ hops in expectation. Substituting $\ell_1 < \ell$, $\ell_2 = \Omega(\ell / \log \ell)$ and $\ell = \log n$, we get $R(n) = O(\log n / \log \log n)$.

Lower bound: R-Hypercube has $\Theta(\log n)$ links per node. For any topology (deterministic or randomized) with $O(\log n)$ links per node, $R(n) = \Omega(\log n / \log \log n)$ hops. $\square$

Theorem 8.6. *$R(n) = \Theta(\log n/\log\log n)$ hops for* GREEDY WITH 1-LOOKAHEAD *routing in randomized Chord.*

Proof. Upper bound: Consider node $\mathbf{x}$ holding a message destined for a node clockwise distance d away. Let $\mathcal{I}$ denote the interval $[d-d'+1, d]$, where $d' = \lceil d\log\log n/\log n\rceil$. Let $2^p \leq d < 2^{p+1}$ such that $p > \log n/\log\log n$. The probability that $\mathbf{x}$ has an out-going edge with a node in interval $\mathcal{I}$ equals $\phi = \frac{a}{2^p} + \frac{b}{2^{p+1}} - \frac{ab}{2^p 2^{p+1}}$ where $a = 2^p - (d-d'+1)$ and $b = d - 2^p + 1$. Simplifying, we get $\phi = \frac{a+b}{2^p} + \frac{a}{2^{p+1}}(1-\frac{b}{2^p}) \geq \frac{a+b}{2^p}$ since $1 \leq b \leq 2^p$. Since $a+b = d'$, we obtain $\phi \geq d'/2^p$. Substituting $d' = \lceil d\log\log n/\log n\rceil$ and $d/2^p \geq 1$, we get $\phi \geq \log\log n/\log n$. Let $\mathbf{u}$ denote a neighbor of $\mathbf{x}$ lying between $\mathbf{x}$ and $\mathbf{x}+d$. The probability of $\mathbf{u}$ having an edge into interval $\mathcal{I}$ is also at least ϕ. There are at least $p > \log n/\log\log n$ such neighbors of $\mathbf{x}$. Therefore, with probability at least $1-e^{-1}$, $\mathbf{x}$ can reach some node in $\mathcal{I}$ in at most two hops. In other words, the expected number of hops to diminish the distance from d to at most $d\log\log n/\log n$ is $O(1)$. Thus in $O(\log d/\log\log n)$ hops, distance d can be diminished to a value less than 2^p where $p < \log n/\log\log n$. The remaining distance requires no more than $O(p)$ hops (since the distance diminishes by a ratio of at least $3/4$ at each hop). Combining the two, expected route length for distance d is $O(\log n/\log\log n)$. Averaged over all values of d, we get $R(n) = O(\log n/\log\log n)$.

Lower bound: R-Chord has $\Theta(\log n)$ links per node. For any topology (deterministic or randomized) with $O(\log n)$ links per node, $R(n) = \Omega(\log n/\log\log n)$ hops. □

Lemma 8.4.1. *The expected length of a* GREEDY *route for covering clockwise distance d in Symphony is $O(\frac{1}{k}\log d\log n)$ hops.*

Proof. We proceed as in the proof of Theorem 7.1. Let p_{half} denote the probability of drawing a value from $[z/2, z]$. For any $z \in [2, n]$, $p_{\text{half}} = \int_{z/2}^{z} p_n(x)dx = 1/\log_2 n$, which is independent of z. The significance of p_{half}: regardless of the current clockwise-distance to the destination, it is the probability that any single long-distance link will cut the distance by at least half. The number of links to consider before the current distance diminishes by at least half follows a geometric distribution with mean $1/p_{\text{half}} = \log_2 n$. With k links per node, the expected number of nodes to consider before the current distance is at least halved is $\lceil(\log_2 n)/k\rceil$, which is less than $(2\log_2 n)/k$ for $k \leq \log_2 n$. The maximum number of times the original distance could possibly be halved before it is less than 2 is $\log_2 d$. Therefore, the expected length of route is at most $2(\log_2 n)(\log_2 d)/k = O(\frac{1}{k}\log n\log d)$ hops. □

Theorem 8.7. *$R(n) = O\left(\frac{\log^2 n}{k \log k}\right)$, when $1 \le k \le \log n$, for* GREEDY *with 1*-LOOKAHEAD *routing in Symphony.*

Proof. If the remaining distance d satisfies $\log d \le \log n / \log k$, then Lemma 8.4.1 assures us that we need no more than $O(\frac{1}{k} \log d \log n) = O(\log^2 n/(k \log k))$ hops using GREEDY.

Let $\mathbf{x}$ denote the node currently holding a message to be shipped to node $\mathbf{y}$, which is clockwise distance d away such that $\log d > \log n / \log k$. Let ratio $r = k/\log k$. We compute the probability that $\mathbf{x}$ is able to forward the message to some node $\mathbf{z}$ in at most two hops such that the clockwise distance from $\mathbf{z}$ to $\mathbf{y}$ is at most d/r. There are three possible ways of reaching such a node $\mathbf{z}$:

(a) Long-distance link $(\mathbf{x}, \mathbf{z})$ exists.

(b) There exists node $\mathbf{u}$ such that both $(\mathbf{x}, \mathbf{u})$ and $(\mathbf{u}, \mathbf{z})$ exist but the clockwise distance from $\mathbf{x}$ to $\mathbf{u}$ exceeds d.

(c) There exists node $\mathbf{u}$ such that both $(\mathbf{x}, \mathbf{u})$ and $(\mathbf{u}, \mathbf{z})$ exist, AND the clockwise distance from $\mathbf{x}$ to $\mathbf{u}$ is at most d.

We will ignore events (a) and (b) above and bound the probability that event (c) occurs from below:

i) Let p_1 denote the probability that a specific long-distance link established by node $\mathbf{x}$ is within distance d. Then $p_1 = \int_1^d 1/(x \ln n) dx = \ln d / \ln n$.

ii) Let $\mathbf{u}$ denote one of the neighbors of $\mathbf{x}$. Let p_2 denote the probability that a specific link of $\mathbf{u}$ connects it to some node $\mathbf{z}$ such that destination $\mathbf{y}$ is at most d/r hops away. If d' denote the distance from $\mathbf{u}$ to $\mathbf{y}$, then $p_2 = \int_{d' - \lfloor d/r \rfloor}^{d'} p(x) dx \ge \int_{d'-d/r}^{d'} p(x) dx \ge \int_{d-d/r}^{d} p(x) dx = \ln(1/(1-1/r))/\ln n > 1/(r \ln n)$.

The probability that event (c) occurs is at least

$$1 - [(1-p_1) + p_1(1-p_2)^k]^k \ge k^2 p_1 / (2r \log n). \tag{8.1}$$

The above inequality can be derived as follows: We know that $(1-p_2)^k < (1 - 1/(r \log n))^k < 1 - k/(2r \log n)$. Now, $1 - [1 - kp_1/(2r \log n)]^k \ge k^2 p_1/(2r \log n)$.

Plugging in $p_1 = \log d / \log n$ into Eq (8.1), we deduce that the probability of event (c) is at least $k^2 \log d/(r \log n)$. We had assumed $\ln d > \log n / \log k$ and we had set the ratio $r = k/\log k$. This implies that the probability of event (c) is at least $ck/\log n$ for some constant c. The expected number of hops required to diminish the current distance by factor r or more is $\lceil (\log n)/ck \rceil$, which is $O((\log n)/k)$ as long as $k = O(\log n)$. For any initial distance, the number of times the remaining distance can be diminished by factor r or more is at most $\log n / \log r$. Thus total path length is $O(\log^2 n/(k \log r))$ hops on average. Since $r = k/\log k$, total path length is $O(\frac{\log^2 n}{k \log k})$ hops on average. □

Theorem 8.8. *$R(n) = O\left(\frac{\log^2 n}{k \log k}\right)$, when $1 \le k \le \log n$, for* GREEDY *with 1-*LOOKAHEAD *routing in Symphony*.*

Proof. Consider node $\mathbf{x}$ that holds a message destined for node $\mathbf{y}$ lying clockwise distance d away. Using a lemma similar to Lemma 8.4.1, we can show that that GREEDY routing takes $O((\log n \log d)/k)$ hops. Therefore, if $\log d \le \log n/\log k$, then the remaining distance can be covered by GREEDY with 1-LOOKAHEAD (which is faster than plain GREEDY) in $O(\log^2 n/(k \log k))$ hops.

We now consider large d satisfying $\log n/\log k < \log d \le \log n$. Let $r(d) = (ck \log d)/\log n$ where d is the clockwise distance currently remaining and c is a constant that we will shortly fix. Since $\log n/\log k < d \le \log n$, we deduce that $ck/\log k < r \le ck$. Let $\mathcal{E}$ denote the event that the current node is able to diminish the remaining distance from d to at most $d/r(d)$ in (at most) two hops. Let $\phi(\mathcal{E})$ denote the probability that event $\mathcal{E}$ occurs. We will shortly prove that $\phi(\mathcal{E}) = \Omega(k/\log n)$, independent of d. Thus the expected number of nodes encountered before a successful event $\mathcal{E}$ occurs is $O((\log n)/k)$. Since $ck/\log k < r$, there can be at most $O(\log n/\log k)$ such events for a total of $O(\log^2 n/(k \log k))$ hops. When d becomes small enough to satisfy $\log d < \log n/\log k$, plain GREEDY routing will take at most $O(\log^2 n/(k \log k))$ hops. Summing the two, the total number of hops is $O(\log^2 n/(k \log k))$.

Proof of $\phi(\mathcal{E}) = \Omega(k/\log n)$: Recall that $\mathcal{E}$ is the event that the current node is able to diminish the remaining distance d to at most $d/r(d)$ in (at most) two hops. Let $d' = \lceil d(1 - 1/r(d)) \rceil$. Let ψ denote the probability that node $\mathbf{x}$ has a link in $[d', d]$. There are at least $k \log d/\log n$ nodes (including $\mathbf{x}$ itself) reachable from $\mathbf{x}$ in zero or one hop, such that the node is at most clockwise distance d away. Let ψ denote the probability that such a node has a link in $[d', d]$. Overall, the probability that one or more of these nodes has a link in $[d', d]$ is $\phi(\mathcal{E}) \ge 1-(1-\psi)^{(k \log d/\log n)}$. We will shortly show that $\psi \ge c'k/(r(d) \log n)$ where c' is some constant. We had defined $r(d) = (ck \log d)/\log n$. We set $c = c'$. This ensures that $(c'k/(r(d) \log n))(k \log d/\log n) = k/\log n \le 1$. Using the fact that $1-(1-x)^t \ge xt/2$

if $x \in (0,1)$ and $xt \leq 1$, we deduce that $\phi(\mathcal{E}) \geq (c'k^2 \log d)/(2r(d)\log^2 n)$. Substituting $r(d) = (c'k \log d)/\log n$, we get $\phi(\mathcal{E}) \geq k/(2\log n)$.

Proof of $\psi = \Omega(k/(r(d)\log n))$: Recall that ψ denotes the probability that node $\mathbf{x}$ has a link in $[d', d]$ where $d' = \lceil d(1 - 1/r(d)) \rceil$. From the definition of δ for Symphony*, $\ln \delta = (\ln n)/k$. If $[d', d]$ is completely contained in some interval $\mathcal{I}_i$ (for definition of $\mathcal{I}_i$, see the definition of Symphony*), then $\psi = s_i^{-1} \sum_{i \in [d', d]} 1/i$. Now $\sum_{i \in [d', d]} 1/i = \Omega(\ln 1/(1 - 1/r(d)) = \Omega(1/r(d))$. Substituting $s_i^{-1} = \Omega(k/\log n)$, we get $\psi = \Omega(k/(r(d)\log n))$. If $[d', d]$ spans two intervals $\mathcal{I}_i$ and $\mathcal{I}_{i+1}$, then $\psi = \psi_1 + \psi_2 - \psi_1\psi_2$, where $\psi_1 = s_i^{-1} \sum_{i \in [d', \delta^i]} 1/i$ and $\psi_2 = s_{i+1}^{-1} \sum_{i \in (\delta^i, d]} 1/i$. Using the fact that $a + b - ab \geq \frac{3}{4}(a + b)$ if $a + b \leq 1$, we deduce that $\psi \geq \frac{3}{4}(\psi_1 + \psi_2)$. Since both s_i^{-1} and s_{i+1}^{-1} are $\Omega(k/\log n)$, we get $\psi \geq \frac{3}{4}(ck/\log n) \sum_{i \in [d', d]} 1/i$, where c is some constant. It follows that $\psi = \Omega((k/\log n) \ln 1/(1 - 1/r(d))) = \Omega(k/(r(d)\log n))$.

When $k = \Theta(\log n)$, $R(n) = \Theta(\log n / \log\log n)$, which is optimal. □

We believe that Chord with $\log n$ short-distance links (with other nodes in the immediate neighborhood of a node along the circle) could also route in $\Theta(\log n/\log\log n)$ hops on average, if we restrict the usage of 1-LOOKAHEAD to long-range contacts of only the set of $\log n$ short-distance neighbors.

8.5 The Bit-Collection Protocol

In this Section, we will study the following two randomized networks, each defined over $n = 2^\ell$ nodes lying on a circle, labeled 0 through $n - 1$. Each node makes a short-distance connection with its successor, and a few long-distance connections randomly:

- ⋆ Sparse-Chord [M03]

  In Chord, a node makes $\ell - 1$ long-distance connections with other nodes at the following clockwise-distances: $\langle \frac{n}{2}, \frac{n}{4}, \frac{n}{8}, \ldots, 4, 2 \rangle$. In Sparse-Chord, each node chooses $k \geq 1$ out of these out-going links at random.

- ⋆ Sparse-Hypercube [M03]

  In a hypercube, a node makes ℓ long-distance connections with other nodes corresponding to bit-flips in each of ℓ positions of its label written in binary. In Sparse-Hypercube, each node chooses $k \geq 1$ out of these out-going links at random.

Theorem 8.9. *It is possible to route in $O(\log n \log\log n)$ hops on average in Sparse-Chord and Sparse-Hypercube with $k = 1$.*

Proof. Let $b = \lceil \log_2 n \rceil$ bits. In Sparse-Chord with $k = 1$, we use a non-GREEDY routing strategy: Let the distance remaining to the destination be d. Let $b' = \lceil \log_2(4b \ln b) \rceil$ bits. If the long link of the current node corresponds to one of the top (most significant) $b - b'$ bit positions where d represented in binary has a 1, then forward the message along the long link. Otherwise, forward the message clockwise along the short link. Forwarding along a long link removes some 1 among the top $b - b'$ bits. The lower order b' bits act as a counter that diminishes by 1 whenever a short link is followed.

The protocol is reminiscent of the classic Coupon Collection problem (see, for example, the book by Motwani and Raghavan [MR95]). Essentially, we have to collect at most $b - b'$ coupons where the probability of collecting a coupon in one step is $1/b$. The expected number of steps required to collect $b - b'$ bits is at most $b \ln b + \Theta(b)$ steps. Building upon this intuition, it can be shown that on average, routing requires $O(b \log b)$ hops. Since $b = O(\log n)$, average route length is $O(\log n \log\log n)$. ☐

Theorem 8.10. *It is possible to route in $O(\log n)$ hops on average in Sparse-Chord and Sparse-Hypercube with $k = \Theta(\log\log n)$.*

Proof. Let $b = \lceil \log_2 n \rceil$ bits. With $k = O(\log b)$ links chosen uniformly out of the b possible, average route length diminishes to $O(\frac{1}{k} \log n \log\log n)$. When $k = \Theta(\log\log n)$, average route length is only $O(\log n)$. ☐

8.6 Related Work

The results in this Chapter have connections with several research areas.

8.6.1 Random Graphs

Random graphs have been studied since 1950s by mathematicians (see books by Bollobás [B01] and Spencer [S01] for theoretical foundations of this area). Traditionally, random graphs have been studied for properties like diameter, connectivity and chromatic number. For example, Bollobás and Chung [BC88] prove that the diameter of a cycle graph augmented with a random matching is small. *Decentralized routing strategies* in random graphs have received attention only recently, beginning with Kleinberg's paper [K00].

8.6.2 Long-range Percolation

In a classical percolation model called "long range percolation", nodes lie on an infinite lattice and an edge exists between a pair of nodes with some positive probability. The question of existence of infinite components was considered by Schulman [S83], Aizenman and Newman [AN86] and Newman and Schulman [NS86], where the one dimensional lattice $\mathcal{Z}$ is studied and edges (i, j) are selected with probability $\beta/\|i-j\|^s$ for some values β, s.

Benjamini and Berger [BB01] proposed and studied a *finite* percolation model: a cycle graph over n nodes where an edge between nodes i and j exists with probability 1 if $\|i-j\| = 1$, otherwise, it exists with probability $\exp(-\beta/\|i-j\|^s)$, for some constants β, s. The norm $\|i-j\|$ denotes the clockwise distance from node i to j (arithmetic is being done modulo n). Coppersmith *et al* [CGS02] extended the model to multiple dimensions. Their model has a $d-$dimensional mesh instead of a one-dimensional ring. Coppersmith *et al* established that the *diameter* of the resulting graph is $\Theta(\log n/\log\log n)$ w.h.p. Their proof used the neighbor-of-neighbor approach for part of the way, and a non-constructive argument for the rest of the way. Thus their proof leaves open the question of whether there exists some efficient decentralized routing algorithm whose performance can match the bound on the diameter asymptotically.

The randomized routing networks that we analyzed in this Chapter share structural similarities with a network in which nodes are associated with a $d-$dimensional torus, and an edge (i, j) is established with probability $\frac{1}{||i-j||^d}$. In reference [MNW04], we call this network a *Small-World Percolation Network*.

> DEFINITION (Small-World Percolation Network [MNW04]). *The probability that an edge (u, v) exists is $1/\delta_{absolute}(u, v)$, where $\delta_{absolute}$ denotes the absolute distance between u and v along the circle. The out-degree of a node is not fixed; it is a random variable. The distance-function for routing is $\delta_{absolute}$.*

In reference [MNW04], it is shown that GREEDY routing in the Small-World Percolation Networks requires $\Theta(\log n)$ hops in expectation, whereas GREEDY with 1-LOOKAHEAD requires only $\Theta(\log n/\log\log n)$ hops on average. In fact, the paper establishes high probability bounds for both results.

The relationship between Symphony and the Small-World Percolation Network is as follows. Symphony is a directed graph. It does not choose each edge independently. Instead, each node chooses k edges independently, with replacement. For each edge, the clockwise distance to the destination is drawn from a probability distribution $p(x) = 1/(x \log n)$ for $x \in [1, n]$. Intuitively, $p(x)$ is similar in behavior to setting $s = 2$ and using the function $\beta/|i-j|^2$ instead of $\exp(-\beta/|i-j|^2)$.

8.6.3 Small-World Models

Random graphs find applications in devising models that possess statistical properties of associations occurring in nature, *e.g.,* social acquaintanceship networks, electric power grids, telephone call graphs, neural wiring of worms and influence networks (see the book by Barabási [B02] for many more examples). The so-called "Small World Models" such as those by Watts and Strogatz [WS98] are characterized by a successful mixture of regularity and randomness to faithfully reproduce three statistical properties: the "characteristic path length", the "average vertex degree" and the "clustering coefficient" (see Newman *et al* [NWS02]). An important property ignored by these models is the existence of short routes, i.e., the *small-world phenomenon.* In this regard, Kleinberg's work [K00] is the first to study *decentralized routing properties* of random graphs.

8.6.4 Kleinberg's Small-World Networks

Decentralized routing algorithms for random graphs have been developed only recently, beginning with Kleinberg's work [K00]. Several results pertaining to Kleinberg's construction with only one long-distance link per node have been obtained: The original paper showed that GREEDY routing requires only $O(\log^2 n)$ hops. Barrière *et al* [BFKK01] showed that GREEDY routing takes $\Omega(\log^2 n)$ steps in 1D, thereby providing a matching lower bound. Martel and Nguyen [MN04] establish the same result using a different proof technique. They also show that the diameter of Kleinberg's construction is $\Theta(\log n)$ in any dimension. They also develop new decentralized routing algorithms which require $O(\log^{3/2} n)$ hops in expectation in 2D and $O(\log^{1+1/d} n)$ hops in d-dimensions for $d \geq 1$. Lebhar and Schabanel [LS04] have developed a different decentralized routing algorithm that requires only $O(\log n \log^2 \log n)$ hops on average. Fraigniaud *et al* [FGP04] study an interesting variant of GREEDY routing in multiple dimensions. Their routing scheme requires $O(\log^{1+1/d} n)$ hops on average in d-dimensions if each node is equipped with $O(\log^2 n)$ bits of "topological awareness". Aspnes *et al* [ADS02] studied a generalization of Kleinberg's construction. They show that if each node makes k long-distance links, each using a common probability distribution function, then average route length is $\Omega(\log^2 n/(k \log\log n))$, no matter which distribution is used.

Symphony [MBR03] is an adaptation of Kleinberg's static routing [K00] network could be adapted to arrive at a randomized P2P topology. Symphony establishes k links per node and routes in $O((\log^2 n)/k)$ hops on average as long as $k \leq \log_2 n$.

Aspnes *et al* [ADS02] studied GREEDY routing in Kleinberg-style networks where each node establishes long-distance links by drawing random numbers drawn from some fixed but unknown probability distribution. They showed that GREEDY requires $\Omega(\log^2 n/(k \log\log n))$ hops on average if each node has k links and each link is used along only one direction. The randomized topologies we study in this paper do not fall into the general class studied by Aspnes *et al* because the long-distance links are drawn from different distributions.

8.6.5 Greedy without Lookahead in Deterministic Networks

Xu *et al* [XKY03] have studied GREEDY routing with distance function $\delta_{clockwise}$ over *uniform* graph topologies. A graph over n nodes placed in a circle is said to be uniform if the set of clockwise offsets of out-going links is identical for all nodes. Chord is an example of a uniform graph. Xu *et al* show that for any uniform graph with $O(\log n)$ links per node, GREEDY routing with distance function $\delta_{clockwise}$ necessitates $\Omega(\log n)$ hops in the worst-case.

Cordasco *et al* [CGH$^+$04] have shown that GREEDY routing with distance function $\delta_{clockwise}$ in a uniform graph over n nodes satisfies the inequality $n \leq F(d + \Delta + 1)$, where d denotes the out-degree of each node, Δ is the length of the longest GREEDY path, and $F(k)$ denotes the k^{th} Fibonacci number. It is well-known that $F(k) = [\phi^k/\sqrt{5}]$, where $\phi = 1.618\ldots$ is the Golden ratio and $[x]$ denotes the integer closest to real number x. It follows that $1.44 \log_2 n \leq d + \Delta + 1$. Cordasco *et al* show that the inequality is strict if $|d - \Delta| > 1$. For the case $|d - \Delta| \leq 1$, they construct uniform graphs based upon Fibonacci numbers which achieve an optimal tradeoff between d and Δ.

Papillon (see reference [AMM04] or Chapter 6) is a non-uniform graph topology that offers GREEDY routes of length $O(\log n/\log d)$ hops when each node makes d out-going links per node.

8.6.6 Viceroy- and Mariposa-style Randomized Networks

The randomized networks that we studied in this Chapter are constructed with the following sequence of operations: (a) Generation of local random bits, (b) Establishment of long-distance links, and (c) Inspection of non-local random bits for routing (*e.g.,* GREEDY with 1-LOOKAHEAD). Viceroy [MNR02] and Mariposa [M03] (see also Chapter 9) are two randomized routing networks that follow a different sequence of operations: (a) Generation of local random bits, (b) Inspection of non-local random bits for establishment of long-distance links, and (c) Routing. Both Viceroy and Mariposa are adaptations of butterfly networks. Viceroy routes in $\Theta(\log n)$ hops with only $\Theta(1)$ out-going links per node. Mariposa has $3\ell + 3$ out-going links per node, and can route in $O(\log n / \log \ell)$ hops in the worst-case, which is asymptotically optimal.

8.7 Discussion

Effect of Randomization upon Route Lengths: Randomization of edges actually *reduces* the average length of shortest paths in Chord and hypercubes. The reason is that the randomization enables a routing algorithm to use an 'exceptionally' long edge once in a while. The density of these long edges is just large enough so that the 1-LOOKAHEAD finds them. In a 'perfect' skip graph, Chord, and in the hypercube - these long edges do not exist. Our results show that safety has a price: while these network topologies have guaranteed worst-case route-lengths, they enlarge the expected length of routes.

System Issues with 1-LOOKAHEAD: An implementation of GREEDY with 1-LOOKAHEAD routing strategy necessitates that each node acquire knowledge of its neighbor's neighbors. At first glance, it might appear that maintenance of such knowledge is problematic since it is tantamount to squaring the degree of the graph and therefore, squaring the size of the routing table at each node. However, it is important to note that the bottleneck in the system is actually the run-time cost of keep-alives exchanged between nodes for maintaining the TCP links between them. This cost remains unchanged, irrespective of which routing protocol we use: GREEDY with/without 1-LOOKAHEAD. Updates to neighborhood sets can easily be piggy-backed on top of the "keep-alive" messages. Also note that the neighbor-of-neighbor information at a node does not have to be perfectly up-to-date at all times to derive the benefits of 1-LOOKAHEAD.

8.8 Summary and Future Work

We studied GREEDY routing with/without 1-LOOKAHEAD in a variety of randomized routing networks: Randomized-Chord, Randomized-Hypercube and Symphony.

Future research directions:

1. It is well-known that de Bruijn graphs and butterfly networks (see the book by Leighton [L92]) can route in $\Theta(\log n / \log\log n)$ hops with $\Theta(\log n)$ links per node. The results of this paper have revealed another family of networks with the same tradeoff – these networks are randomized. In Chapter 9, we will describe yet another family of randomized networks with the same tradeoff – the new family has a different philosophy underlying its construction. We hope that our results inspire further investigations into the general properties of these randomized/deterministic networks, and the relationships among them.

2. We suspect that when GREEDY WITH 1-LOOKAHEAD routing is employed in Symphony/Symphony*, the average route length is $\Omega(\log^2 n/(k \log k))$ hops.

3. Aberer [A02] has proved an interesting result in the context of P-Grid [ACMD+03]. He considers the scenario where nodes choose their IDs from $[0, 1)$ in an *adversarial* fashion. Now if we construct a randomized hypercube (see [A02] for the precise definition used by Aberer since it is different from our definition of Randomized-Hypercube), then the average route length is $O(\log n)$ hops. It might be interesting to extend results pertaining to GREEDY routing to adversarial choices of IDs.

4. What is the diameter of Symphony for general values of k, the number of long-distance links established by every node? When $k = \Theta(\log n)$, we showed that the diameter is $\Theta(\log n/ \log\log n)$. For general k, tight bounds on the diameter are not known.

Chapter 9

Mariposa: A Randomized Butterfly

In this Chapter, we describe Mariposa, a randomized routing network whose design philosophy differs from the randomized networks we studied in Chapters 7 and 8. The difference lies in whether a node learns about the random choices made by other nodes *before* or *after* link-establishment. We explain the point in more detail below.

In Randomized-Chord [ZGG03,GGG$^+$03], Randomized-Hypercube [CDHR03,GGG$^+$03], Symphony (see reference [MBR03] or Chapter 7), SkipNet [HJS$^+$03], skip-graphs [AS03] and Small-World Percolation Networks [MNW04], the routing network is constructed as follows. Each node generates some random bits locally, and establishes links with other nodes on the basis of the bits it generates. *After* link-establishment, a node can inspect the random bits of other nodes (typically, its neighbors with whom it has established links) for making good routing decisions. For example, "GREEDY with 1-LOOKAHEAD routing" (studied in Chapter 8) follows this paradigm. In Mariposa, each node first generates some random bits locally. It then inspects the random bits of a few other nodes *before* it establishes its long-distance links. The knowledge gained by inspecting other nodes' random bits is used for making good decision for link-establishment itself.

In a nutshell, all routing networks in Chapter 8 used the following sequence of operations:

- ➢ Generation of local random bits.
- ➢ Establishment of long-distance links.
- ➢ Inspection of non-local random bits for routing.

Mariposa uses the following sequence of operations:

- ➢ Generation of local random bits.
- ➢ Inspection of non-local random bits for establishment of long-distance links.
- ➢ Routing.

Mariposa is an interesting combination of butterfly networks and Kleinberg's small world construction [K00]. With $3\ell + 3$ out-going links per node, worst-case route lengths are $O(\log n/\log \ell)$ hops with high probability†, which is asymptotically optimal. Mariposa improves upon Viceroy [MNR02], an earlier butterfly-based construction that routes in $O(\log n)$ hops in expectation with $\Theta(1)$ links per node.

From a systems standpoint, Mariposa and Viceroy are complex constructions. However, understanding them yields insights into the design space of randomized routing networks, and sheds light on the algorithmic relationships between various randomized and deterministic routing networks.

Summary of Results

In §9.1, we describe the construction of Mariposa.

In §9.2, we prove the $O(\log n/\log \ell)$ bound for the worst-case routes in Mariposa.

In §9.4, we summarize and present directions for further research.

9.1 Mariposa: the Construction

Mariposa‡ is constructed over n nodes lying on the circumference of a circle. We will deal with two distributions:

- ⋆ Random Distribution: Each node has chosen its position independently and uniformly at random on the circumference of the circle.
- ⋆ Regular Distribution: The n nodes occupy positions corresponding to the corners of a regular n-gon circumscribed by the circle.

Each node in Mariposa maintains three real numbers:

† By "with high probability" (w.h.p.), we mean "with probability at least $1 - O(n^{-\lambda})$ for some constant $\lambda > 1$, for a system with n participants".

‡ Mariposa means butterfly in Spanish.

✧ Position p

Each node will be assigned a position lying in the unit interval $\mathcal{I} = [0, 1)$. It is convenient to imagine $\mathcal{I}$ as a circle with unit perimeter. The binary operators $+$ and $-$ wrap around the interval $\mathcal{I}$. In other words, $x + y$ denotes the point that lies clockwise distance y away from x along the circle. Similarly, $x - y$ denotes the point that lies anti-clockwise distance y away from x.

For a random distribution of nodes, position p is chosen uniformly at random from $\mathcal{I}$. For a regular distribution of nodes, position p is a real number in $\mathcal{I}$, corresponding to one of the vertices of a regular n-gone circumscribed by the circle.

✧ Estimate $\tilde{n}$

For random distribution of nodes, in Chapter 4, we had described a distributed Network Size Estimation procedure by which a node can estimate the number of nodes as $\tilde{n}$, with the guarantee that $\tilde{n} \in [\frac{1}{4}n, 4n]$. Such an estimate can be made by inspecting the positions of a few adjacent nodes along the circle. We re-state this result as Lemma 9.2.1 in Section 9.2.

For regular distribution of nodes, $\tilde{n} = n$.

✧ Range r

A node chooses as its range r, a real number drawn from a *range probability distribution*

$$\mathcal{P}_{\tilde{n}} = 1/(x \ln \tilde{n}) \qquad \text{for } x \in [1/\tilde{n}, 1]$$

Distribution $\mathcal{P}_{\tilde{n}}$ is simply the continuous version of the discrete distribution in Kleinberg's paper [K00]. A node at position p with range r is said to *span* the interval $[p - r, p] \cup [p, p + r]$. Note that $[p - r, p]$ and $[p, p + r]$ could have more than one point in common if $r \geq 0.5$.

Hereafter, we will not make a distinction between random and regular distribution of nodes. We will assume that each node maintains $\tilde{n}$, an estimate of the total number of nodes. By setting $\tilde{n} = n$, we obtain the construction over a regular distribution of nodes.

Each node maintains $3\ell + 3$ outgoing links where $\ell \geq 1$. We will assume that $\ell = O(\text{polylog}(n))$. For $\ell \geq 2$, a node establishes 1 *short link*, 2 *intermediate links*, 2ℓ *long links* and at most ℓ *global links*. When $\ell = 1$, a node maintains 1 short link, 1 intermediate link, 2 long links and at most 2 global links. In any case, the total number of links is $3\ell + 3$ for

$\ell \geq 1$.

Short and Intermediate Links

A short link is established with the clockwise successors of a node. For $\ell \geq 2$, intermediate links are established with two nodes that are $\lceil \log \tilde{n} \rceil$ and $\lceil \log \tilde{n} / \log \ell \rceil$ hops away in the clockwise direction along the circle. When $\ell = 1$, only one intermediate link is established with the node that is $\lceil \log \tilde{n} \rceil$ hops away in the clockwise direction.

Short and intermediate links are used to route when the target is known to be nearby. Lemma 9.2.3 will show that a node that is $O(\log^2 n / \log \ell)$ hops away is reachable in only $O(\log n / \log \ell)$ steps.

Long Links

Long links lie at the heart of our protocol. A node at position p with range r partitions the interval $[p - r, p]$ into ℓ non-overlapping equi-sized sub-intervals and establishes one long link per sub-interval. It establishes ℓ additional links by partitioning the interval $[p, p + r]$ into ℓ non-overlapping equi-sized sub-intervals. Note that $[p - r, p]$ and $[p, p + r]$ would have more than one point in common if $r \geq 0.5$.

Let I_{sub} denote one of the sub-intervals of $[p - r, p]$ or $[p, p + r]$. By definition, its size is $|I_{sub}| = r/\ell$. Let p_{sub} denote the mid-point of I_{sub}. We now define an interval I_{search} with $|I_{search}| = 64 \ln^2 \tilde{n} / (\tilde{n} \ln \ell)$, centered at p_{sub}. Note that $|I_{search}|$ is independent of r. If $|I_{search}| \geq |I_{sub}|/2$, we say that I_{sub} is a *small* sub-interval. Otherwise I_{sub} is said to be a *large* sub-interval. If I_{sub} is small, we establish a link with the manager of the point $p_{sub} - r/(2\ell)$. If I_{sub} is large, we invoke a routine called SEARCH. The goal of SEARCH is to discover some node whose position is within I_{search} and whose range lies within the interval $[3r/(4\ell), 7r/(8\sqrt{\ell})]$. It is easy to see with $|I_{search}| < |I_{sub}|/2$, it is guaranteed that the span of such a node would include every point of I_{sub}. As Lemma 9.2.5, we will prove that w.h.p., all invocations of SEARCH succeed. This is because the interval I_{search} is sufficiently large in size.

For a node at position p with range r, we claim that all points within $[p-r, p] \cup [p, p+r]$ are reachable by short paths. To reach the manager of some point, we identify the sub-interval to which the point belongs and forward the lookup along that long link that corresponds to this sub-interval. If the sub-interval is small, we arrive at a node such that the destination is no more than $64 \ln^2 \tilde{n} / (\tilde{n} \ln \ell)$ away. At this point, intermediate and short links can carry out further routing. Lemmas 9.2.2 and 9.2.3 will show that this requires no more than $O(\ln n/\ln \ell)$ steps. If the sub-interval is large, we arrive at a node whose range is at most $7r/(8\sqrt{\ell})$. The idea is that shrinking by a factor of $7/(8\sqrt{\ell})$ limits the number of long links along any path to $O(\ln n/\ln \ell)$. We will prove our claims formally in Section 9.2.

One aspect of our construction remains. A lookup request can originate at a node that does not include the destination in its span. This might happen if $r < 0.5$. In such a case, how do we reach a node with range large enough to include the destination? Global links solve this problem.

Global Links

Global links are established by a node with range $r < 0.5$. Consider $I - [p-r, p+r]$ where I denotes the full circle. For $\ell \geq 2$, we partition the interval $I - [p-r, p+r]$ into ℓ equi-sized sub-intervals having size $(1-2r)/\ell$ each. For each sub-interval I_{sub}, we invoke SEARCH with the size and location of I_{search} being similar to our earlier description for long link establishment. The only change is that SEARCH looks for a node with range lying in the interval $[3(1-2r)/(4\ell), 1]$. When $\ell = 1$, we partition $I - [p-r, p+r]$ into two equi-sized sub-intervals with size $(1-2r)/2$ each. SEARCH is invoked twice to look for a pair of nodes, one in each sub-interval, with ranges lying in $[3(1-2r)/8, 1]$.

When a node initiates a lookup request, it forwards it along that global (or local) link whose span includes the destination point. Thereafter, the request is forwarded along a series of long links until we reach a sub-interval that is small. Hereafter, intermediate and short links are used for routing.

9.2 Analysis

Lemma 9.2.1. *With probability at least* $1 - 2/n$, *all nodes in a network of size* n *have* $\tilde{n} \in [\frac{1}{4}n, 4n]$, *assuming a random distribution of nodes.*

Proof. From Theorem 4.2 (for sufficiently small δ). $\square$

We will establish that w.h.p., the worst case route length is $O(\ln n/\ln \ell)$ for $\ell = O(\text{polylog}(n))$. The overall proof idea is as follows. First, we show that small sub-intervals do not have high densities (A sub-interval is *small* if its size is less than $64\ln^2\tilde{n}/(\tilde{n}\ln \ell)$. In particular, we will show that w.h.p., no small sub-interval has more than $O(\ln^2 n/\ln \ell)$

nodes. Next, we will establish that with probability at least $1 - 3\ell/n$, all invocations of Search succeed. The resulting topology enjoys the property that path lengths of lookups are guaranteed to be as small as $O(\ln n/\ln \ell)$. Overall, we would have proved that w.h.p., the worst case route length for a lookup is $O(\ln n/\ln \ell)$.

Lemma 9.2.2. *With probability at least $1 - 2/n$, no small sub-interval has more than $O(\ln^2 n/\ln \ell)$ nodes.*

Proof. Using Chernoff Inequality and Lemma 9.2.1, we can show that with probability at least $1-2/n^2$, a particular sub-interval cannot be dense. Summing over all nodes, we obtain the requisite bound. □

The role of intermediate links is to route quickly to any node that is $O(\ln^2 n/\ln \ell)$ hops away along the circle.

Lemma 9.2.3. *Intermediate and short links can be followed to reach any node that is $O(\ln^2 n/\ln \ell)$ hops away in the clockwise direction in $O(\ln n/\ln \ell)$ steps.*

Proof. The longer of the two intermediate links can be followed in succession to reach a node that is at most $O(\ln n)$ hops away. This requires $O(\ln n/\ln \ell)$ steps. Then the shorter of the intermediate links can be followed to reach a node within $O(\ln n/\ln \ell)$ hops of the destination. This requires $O(\ln \ell)$ steps. Finally, $O(\ln n/\ln \ell)$ short links can be followed to reach the destination. Since $\ell = O(\text{polylog}(n))$, the total number of steps is $O(\ln n/\ln \ell)$. □

For small sub-intervals, long and global link establishment always succeeds. If the sub-interval is large, there is a chance that Search fails.

Lemma 9.2.4. *An invocation of* Search *fails with probability at most $1/n^2$.*

Proof. We will prove the lemma for long links. The proof for global links is along the same lines.

Search is invoked only if $|I_{sub}|/2 > |I_{search}|$. This implies $r/2\ell > 64 \ln^2 \tilde{n}/(\tilde{n} \ln \ell)$, where $\tilde{n}$ is the estimate of the node that invoked Search. Thus, $3r/(4\ell) > 96 \ln^2 \tilde{n}/(\tilde{n} \ln \ell)$ $> 16/\tilde{n}$ for large n. From Lemma 9.2.1, $16/\tilde{n} \geq 4/n$, which is definitely larger than $1/\tilde{n}$ for any node in I_{search} being probed.

When establishing long links, the goal of Search is to discover some node whose range lies in $[3r/(4\ell), 7r/(8\sqrt{\ell})]$. The probability that the range of a node with estimate $\tilde{n}$ lies in this interval is given by $p = \int_{3r/4\ell}^{7r/8\sqrt{\ell}} 1/(x \ln \tilde{n})dx$. In the preceding paragraph, we showed that $3r/4\ell \geq 1/\tilde{n}$ for any node in I_{search}. Therefore, the value of the integral is $(\ln \frac{7}{6}\sqrt{\ell})/\ln \tilde{n}$. From Lemma 9.2.1, this quantity is at least $(\ln \frac{7}{6}\sqrt{\ell})/(2\ln n)$.

$|I_{search}| = 64 \ln^2 \tilde{n}/(\tilde{n} \ln \ell)$. Lemma 9.2.1 yields $|I_{search}| \geq 8 \ln^2 n/(n \ln \ell)$ for large n.

Let us fix the position of the node which invoked Search. Consider the sequence of $n-1$ remaining nodes choosing their positions and ranges one by one. With probability $1 - |I_{search}|$, the position does not lie in I_{search}. Otherwise, with probability at least $(\ln \frac{7}{6}\sqrt{\ell})/(2\ln n)$, Search succeeds. Thus the probability that no node makes Search succeed is at most $[1 - |I_{search}| + |I_{search}|(1 - \frac{\ln \frac{7}{6}\sqrt{\ell}}{2\ln n})]^{n-1} \leq [1 - \frac{(8\ln^2 n)(\ln \frac{7}{6}\sqrt{\ell})}{(n\ln \ell)(2\ln n)}]^{n-1} \leq [1 - \frac{2\ln n}{n}]^{n-1} = o(1/n^2)$. □

Lemma 9.2.5. *With probability at least $1 - 3\ell/n$, all invocations of* Search *succeed.*

Proof. Lemma 9.2.4 shows that the probability that a particular invocation of Search fails is at most $1/n^2$. Since there are at most $3\ell n$ total invocations, the probability that *any* of them fails is at most $3\ell/n$. □

The next lemma shows that although we allow Search to probe all nodes in a rather large sized I_{search}, the expected number of nodes it needs to probe is much smaller.

Lemma 9.2.6. Search *probes an average of $O(\ln n/\ln \ell)$ nodes before succeeding.*

Proof. In the proof of Lemma 9.2.5, we proved that the probability that a node in I_{search} makes Search invoked for long links succeed is at least $(\ln \frac{7}{6}\sqrt{\ell})/(2\ln n)$. Thus the expected number of nodes to be probed before we succeed is at most $(2\ln n)/(\ln \frac{7}{6}\sqrt{\ell}) = O(\ln n/\ln \ell)$. The same bound can be established for Search invoked for global links. □

We proved that w.h.p., all global and long links successfully get established. The resulting topology enjoys the property that all lookup paths are short.

Theorem 9.1. *With probability at least $1 - (6 + 3\ell)/n$, the worst case route length is $O(\ln n/\ln \ell)$ hops, when $\ell = O(\text{polylog}(n))$.*

Proof. From Lemmas 9.2.1, 9.2.2 and 9.2.5, we conclude that with probability at least $1 - (6 + 3\ell)/n$, all estimates of n are within a factor of four, no small sub-interval is dense and all long and global links get established. We show that the resulting graph has short diameter.

Routing proceeds in two phases. In the first phase, a lookup is forwarded along some long or global link whose range is guaranteed to contain the destination. The request then moves along a series of long links such that every node along the path has a range large enough to contain the destination in its span. The first phase starts at some node with range at most 1. From 9.2.1, when the first phase finishes, the last node will have range at least $1/4n$. Since each long link along the path shrinks the range by at least $\frac{8}{7}\sqrt{\ell}$, the first phase requires no more than $O(\ln n/\ln \ell)$ hops.

The second phase starts when we encounter a node with tiny range such that all its sub-intervals are small. At this point, the destination is only $O(\ln^2 n/(n \ln \ell))$ hops away (Lemma 9.2.2). Intermediate and small links can reach the destination in $O(\ln n/\ln \ell)$ steps (Lemma 9.2.3).

Total route length is thus $O(\ln n/\ln \ell)$ w.h.p. □

Reducing the Out-degree

We briefly outline a construction that requires only 4 links per node for $O(\ln n)$ average route length w.h.p. We set $\ell = 1$ and get rid of global links. Note that a fraction $\approx (c/\ln n)$ nodes will have their range smaller than c'/n for some constants c and c'. These nodes will not establish long links since their range is tiny. They will instead establish two global links each. Routing now requires that a lookup be forwarded to some node with tiny range. Hereafter, the usual protocol works. We can reduce the number of links to only 3 per node by removing the intermediate link as well. The resulting topology has an average route length of $O(\ln n/\ln \ell)$. However, the high probability bound no longer holds.

9.3 Intuition

In this Section, we develop intuition underlying the design of several routing networks: Chord, Kleinberg's construction [K00], Symphony [MBR03], sparse-Chord (studied in Chapter 8), Viceroy [MNR02], and Mariposa. We attempt to show that these networks constitute a continuum of design choices with Chord and Mariposa lying at two extremes.

Consider a cycle graph on n nodes where vertices are labeled $0, 1, 2, \ldots, n-1$ and there is an edge between node i and node $(i+1) \bmod n$. A message can be routed clockwise from a node to any other in at most $n-1$ steps. By the introduction of a few more links per node, routes can be made shorter.

Assume that a message destined for node $\mathbf{x_{dest}}$ is currently in possession of node $\mathbf{x_{src}}$. Let $d = (n + \mathbf{x_{dest}} - \mathbf{x_{src}}) \bmod n$, the distance between the nodes. Let h denote the number of 1's in $\mathbf{x_{dest}} \oplus \mathbf{x_{src}}$, the Hamming distance between the two nodes. With the exception of de Bruijn graphs, there seem to be two fundamental themes lying at the heart of existing routing protocols: A route diminishes either the distance d or the Hamming distance h to the destination. CAN [RFHK01], Chord [SMK$^+$01], Kleinberg's construction [K00], Symphony [MBR03], Viceroy [MNR02] and Mariposa [M03] are designed with d in mind. Hypercubes, Pastry [RD01a] and Tapestry [ZHS$^+$04] are designed with h in mind. Routes that diminish d do not necessarily diminish h and vice versa. However, the intuition behind both flavors of routing has commonalities, e.g., a protocol gradually diminishes the number of 1's in either d or h. We now present a unified picture of routing networks that diminish distance d during routing.

Distance Halving

Consider the function $\mathcal{C}_n(x) = (\ln nx)/\ln n$ for $x \in [\frac{1}{n}, 1]$. This is the cumulative probability distribution of $\mathcal{P}_n(x) = 1/(x \ln n)$ for $x \in [\frac{1}{n}, 1]$. For $x \in [\frac{1}{n}, 1]$, we will say that its *notch value* is $y = \mathcal{C}_n(x)$. While routing, let the current distance to the destination be $x_{current}$ with notch value $y_{current}$. Let $s = 1/\log_2 n$. If the current node has a link with notch value between $y_{current} - s$ and $y_{current}$, then we can forward the lookup along this link such that $x_{current}$ is at least halved and $y_{current}$ diminishes by at least s. The maximum number of times $x_{current}$ can be halved (and $y_{current}$ diminished by s) is at most $1/s = \log_2 n$. This intuition underlies several routing protocols that diminish distances.

1. The Chord routing network corresponds to every node establishing exactly $\log_2 n$ links corresponding to notch values $\langle 1-s, 1-2s, 1-3s, \ldots \rangle$. When a node wishes to route to a point $x_{current}$ away (with notch value $y_{current}$), it can immediately forward the lookup along a link such that $x_{current}$ is at least halved and $y_{current}$ diminished by at least $s = 1/\log_2 n$. The worst-case route length is thus $O(\log n)$.

2. In Kleinberg's construction [K00], each node establishes one long link with another node at a distance drawn from a discrete distribution which is quite similar to $\mathcal{P}_n$. This is equivalent to choosing a notch value uniformly at random from $[0, 1]$. Routing proceeds clockwise greedily. If the long link takes us beyond the destination, the request is forwarded to a node's successor. Otherwise, the long link is followed. Let

us denote the current distance to the destination by $x_{current}$ with notch value $y_{current}$. With probability $s = 1/\log_2 n$, the long link of the current node has notch value lying between $y_{current} - s$ and $y_{current}$. Thus the expected number of nodes that need to be visited before we arrive at a node which halves $x_{current}$ is $1/s = \log_2 n$. Effectively, in comparison with Chord, there is an inflation in average route lengths by a factor of $O(\log n)$. Kleinberg's routing scheme requires $O(\log^2 n)$ steps.

3. Symphony extends Kleinberg's idea in the following way. Instead of one long-distance per node, there are k long-distance links where $k \leq \log_2 n$. Effectively, a node gets to choose k notches uniformly from $[0, 1]$. Loosely speaking, when we are at $x_{current}$ (with notch value $y_{current}$), we need to examine roughly $(\log_2 n)/k$ nodes before we encounter some link that diminishes $x_{current}$ by at least half. Thus average route length for Symphony is $O(\frac{1}{k}\log^2 n)$ hops.

4. Sparse-Chord was studied in §8.5, and is defined as follows: Consider a cycle graph on n nodes. Let $b = \lceil \log_2 n \rceil$ bits. A node with ID $\mathbf{x}$ chooses an integer r uniformly at random from the set $\{1, 2, \ldots, b\}$ and establishes a link with node $\lceil \mathbf{x} + n/2^r \rceil \bmod n$.

   It is possible to route clockwise in $\Theta(\ln n \ln \ln n)$ steps in expectation by using a non-greedy decentralized routing strategy, as follows. Let the distance remaining to the destination be d. Let $b' = \lceil \log_2(4b \ln b) \rceil$ bits. If the long link of the current node corresponds to one of the top (most significant) $b - b'$ bit positions where d represented in binary has a 1, then forward the message along the long link. Otherwise, forward the message clockwise along the short link. Forwarding along a long link removes some 1 among the top $b - b'$ bits. The lower order b' bits act as a counter that diminishes by 1 whenever a short link is followed.

   The protocol is reminiscent of the classic coupon collection problem [MR95]. Essentially, we have to collect at most $b - b'$ coupons where the probability of collecting a coupon in one step is $1/b$. It is well known that all $b - b'$ bits can be collected in $2b \ln b$ steps in expectation. Building upon this intuition, it can be shown that on average, routing requires $O(b \ln b)$ hops. Since $b = O(\ln n)$, average latency is $O(\ln n \ln \ln n)$.

   With $\ell \leq \ln b$ links chosen uniformly out of the b possible, it can be shown that average latency diminishes to $O((\ln n \ln \ln n)/\ell)$. With $\ln \ln n$ links, average latency is only $O(\ln n)$. For large values of ℓ, a further improvement is possible. The key idea is that ℓ links can be used to fix $\lfloor \ln_2 \ell \rfloor$ bits in one hop. It can be shown that for large ℓ, routing requires $O((\ln n/\ln \ell)\ln(\ln n/\ln \ell))$ hops.

5. Viceroy: Before we describe Viceroy, let us investigate the Bit-Collection protocol further. Bit-Collection is only a factor $O(\log(\log n/\log \ell))$ more expensive than the best possible protocol. How could we possibly make it faster? By chaining the bits being collected. We illustrate the idea for a network with n nodes. Consider a node $\mathbf{x}$ with a finger that should point to $\lceil \mathbf{x} + n/2^r \rceil \bmod n$ for some integer r. This finger fixes the r^{th} most significant bit. If we could make it point to a node that fixes the $(r+1)^{th}$ bit, then we could hope to collect bits rapidly in succession. The key idea is to search for a pair of nodes, one each in the vicinity of $\mathbf{x}$ and $\mathbf{x} + n/2^r$ that both fix the $(r+1)^{th}$ bit. The two searches on average require only b steps each. How would routing work? If $\mathbf{x}$ wishes to send a message to some node, we first search for a node in the vicinity of $\mathbf{x}$ that fixes the top bit. This requires b steps on average. Then, routing proceeds rapidly by fixing successive top-order bits. A problem that emerges is that searches associated with the top order bits collectively introduce a bias of roughly $O(b^2)$. If every node maintains an additional pointer that points a fixed distance b away, the last stretch of length $O(b^2)$ can be covered in only $O(b)$ steps.

   The intuition developed in the previous paragraph is exactly how Viceroy [MNR02] would work if all nodes knew n precisely (that is, for a Regular distribution of IDs). Using the terminology of notches developed earlier in this Section, Viceroy assigns each node a notch value drawn uniformly at random from the set $\{1-s, 1-2s, 1-3s, \ldots\}$. The size of the set is $\log_2 n$. The relationship with Chord is the following. A Chord node uses the entire set for link establishment resulting in $\log_2 n$ links per node. However, a Viceroy node at position $p \in [0,1)$ and notch value y (corresponding to distance $x = \mathcal{C}_n^{-1}(y)$), searches intervals centered around points p and $p+x$ for a pair of nodes with notch value $y - s$.

6. Mariposa goes a step further than Viceroy. Instead of being restricted to a finite number of discrete values for notch values, Mariposa chooses a notch-value in $[0,1)$ uniformly at random. Moreover, Mariposa uses $3\ell + 3$ links per node as against $\Theta(1)$ links used by Viceroy.

9.4 Summary and Future Work

We described Mariposa, a combination of butterfly networks and Kleinberg's small world construction [K00] that routes in $O(\log n/\log \ell)$ hops with only $3\ell + 3$ links per node. The construction is an improvement over Viceroy [MNR02] which routes in $O(\log n)$ hops on average with only $\Theta(1)$ links per node. As future work, it would be interesting to understand further the relationships between randomized and deterministic routing networks.

Chapter 10

Summary

A Distributed Hash Table (DHT) is a giant hash table that is cooperatively maintained by a large number of machines worldwide. The machines join and leave the system autonomously. The unprecedented scale and dynamism of the system calls for novel design techniques, with emphasis on decentralization and automatic re-configuration. Briefly, each machine in a DHT – a host on the Internet – is assigned an ID in $\mathcal{I} = [0, 1)$. The set of IDs divides $\mathcal{I}$ into disjoint partitions, managed by one machine each. As a function of their IDs, the machines set up connections among themselves. These connections are used for routing messages between the machines. The challenge lies in devising efficient *decentralized* algorithms for ID management and connection maintenance.

10.1 Our Contributions

In this thesis, we presented Dipsea, a modular architecture for building Distributed Hash Tables (DHTs). A block-diagram for Dipsea is shown in Figure 10.1 on the next page. The design consists of three layers: ID Management, Overlay Routing Layer and Data Management – this thesis focuses on the first two layers. The modularity of the design imbues the overall system with several good properties. A large complex problem is broken down into smaller sub-problems, each of which can be attacked more or less independently. This contributes to reduction in complexity — it is possible to explore the design space of a sub-problem in its entirety without being encumbered by its interactions with other sub-problems. Furthermore, the best solutions for individual sub-problems can be identified and put together to arrive at a design that is far more powerful than a design arrived at by

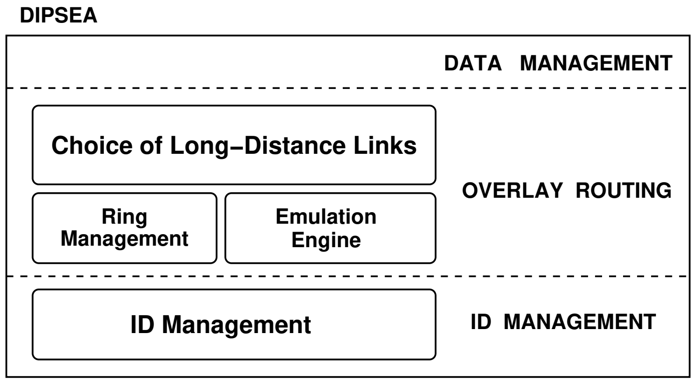


Figure 10.1: Dipsea: *A three-layered architecture for building Distributed Hash Tables. Efficient algorithms for ID Management are described in Chapters 2 and 3. See Section 1.3 for a brief description of Ring Management. See Chapter 4 for the Emulation Engine, and Chapters 5 through 9 for Choice of Long-Distance Links. Data Management is not discussed in this dissertation.*

a holistic approach.

Dipsea places existing DHT designs and improvements suggested for various DHTs into a common algorithmic framework. A significant accomplishment is the identification of layers and modules which are cleanly separated on the basis of functionality. Then for each module, we devise and analyze efficient algorithms – almost all of the algorithms we propose are the currently best-known algorithms for the corresponding modules.

A brief summary of our contributions in ID Management and Overlay Routing layers:

1. In Chapter 2, we presented a simple, decentralized ID Management algorithm which is independent of the Overlay Routing layer. The algorithm requires $O(R + \log n)$ messages and only one re-assignment of existing IDs in response to arrival or departure of machines. R denotes the average number of messages required by the Overlay Routing layer, and n denotes the current number of machines in the system. The algorithm guarantees that the ratio of the largest to the smallest partition is $\Theta(1)$.

2. In Chapter 3, a generalization of the above scheme was presented. The analysis of the generalized algorithm requires the solution to a novel Structured Coupon Collection Problem over cliques with multiple-choices per trial. Using the generalized algorithm, it is possible to carry out ID assignment in $O(aR + b)$ messages, for any choice of a and b satisfying $ab \geq c \log n$, where c is a suitably large constant.

3. In Chapter 4, we presented the Emulation Engine which can emulate or track arbitrary families of deterministic and randomized routing networks. This enables users of Dipsea to plug-and-play any family of routing networks. This is in sharp contrast with first-generation DHT designs like CAN [RFHK01], Chord [SMK+01], Pastry [RD01a] and Tapestry [ZHS+04], which are tied to specific families of routing networks. The Emulation Engine successfully addresses issues pertaining to dynamism (arrivals and departures of hosts), scale (variation in the total number of hosts in the system) and physical network proximity (almost all the links should be low-latency). Crucially, these issues are addressed *independent* of the family of routing networks being emulated, in a generic fashion.

4. In Chapters 5 through 9, we designed and analyzed a variety of deterministic and randomized routing networks. Chapter 5 contains a characterization of shortest paths in Chord, a deterministic DHT routing network.

5. In Chapter 6, we designed Papillon, a butterfly-based graph defined over n nodes placed in a circle. Papillon supports efficient greedy routing, in which each node forwards a message along that out-going edge which reduces the clockwise distance to the destination by the largest amount. Papillon routes in $O(\log n/\log d)$ hops with d links per node, which is asymptotically optimal. This is the first known construction that provides this tradeoff.

6. In Chapter 7, we designed Symphony, one of the first randomized routing networks proposed for DHT routing. With k links per node, clockwise-greedy routing takes $O(\frac{1}{k}\log^2 n)$ hops on average.

7. In Chapter 8, we presented a tight analysis of clockwise-greedy routing with/without lookahead in several randomized routing networks including Symphony, randomized Chord, and randomized hypercube. The idea underlying "greedy with lookahead" is to allow a node to use knowledge of its neighbor's neighbors for better routing decisions. We showed that greedy routing without lookahead requires $\Theta(\log n)$ hops whereas greedy with lookahead entails only $\Theta(\log n/\log\log n)$ hops. Empirically, the average route length with lookahead is within 10% of the average route length in the best-known deterministic constructions.

8. In Chapter 9, we presented Mariposa, an interesting combination of butterfly networks

and Kleinberg's small-world construction. Mariposa also offers routes of $O(\log n/ \log d)$ hops in the worst case, with d out-going links per node. The design philosophy underlying Mariposa is different from that of randomized networks analyzed in Chapter 8. The key difference is whether a node inspects the random choices made by other nodes *after* or *before* it establishes its links.

10.2 Directions for Further Research

At the end of each Chapter, we outlined directions for further research related to the problem we addressed in that Chapter. Below are outlined some high-level issues that merit further investigation.

1. *Heterogeneity*: The design of Dipsea assumes that all hosts are homogeneous, i.e., identical in terms of resources they possess. Clearly this assumption is not true in the real world. It would be interesting to extend Dipsea to incorporate heterogeneity of hosts. See Godfrey and Stoica [GS04] for initial results along this direction.

2. *Data Management*: As new applications are built atop DHTs, we expect the requirements of application-specific Data Management modules to influence the design of the Overlay Routing layer.

3. *Locality*: In a hierarchical name-space for distributed objects, real-world computations exhibit locality of reference. A DHT name-space is flat since object-names are hashed – locality is lost. One possible technique of introducing locality in a DHT is to limit name-resolution to internal nodes of the hierarchy which are suitably far away from the leaves (so that leaves below the name that was resolved are co-located at some server). Efficient schemes for managing such name-resolution in a dynamically changing set of distributed objects requires further investigation.

4. *Caching*: DHTs are useful for caching a large number of objects at a global scale, where the replication factor of each object is proportional to its demand. One way to achieve this goal is to cache an object along the route taken by a query to retrieve the object. See the design of Tapestry [ZHS$^+$04], CUP [RB03] or Beehive [RS04a], for example. It would be interesting to build mathematical models for replication strategies and to analyze them formally.

Appendix A

Proof of Lemma 2.3.1

Definitions from Section 2.3:

$$
\begin{array}{rcll}
\phi(\ell) & = & \max\{0, \ell - \lceil \log_2 \ell \rceil - c\} & \\
\chi(\ell) & = & 1 & \text{if} \quad \ell = 0 \\
 & & 2^{\ell-1} & \text{if} \quad \phi(\ell) = 0 \ \text{ but } \ell \neq 0 \\
 & & 2^{\lceil \log_2 \ell \rceil + c - 1} & \text{otherwise} \\
Y(\ell) & = & \displaystyle\sum_{i=0}^{\ell} G_{\chi(i),\phi(i)} &
\end{array}
$$

where $G_{\chi(i),\phi(i)}$ denote the sum of $\chi(i)$ geometric random variables, each with probability parameter $2^{-\phi(i)}$. Its expectation is $\mathbf{E}Y(\ell) = \sum_{i=0}^{\ell} \chi(i) 2^{-\phi(i)} = 1 + \sum_{i=1}^{\ell} 2^{i-1} = 2^{\ell}$.

A.1 Proof of Lemma 2.3.1(a)

Lemma A.1.1. *For all values of t and p,*

$$[\frac{2pe^t}{1-(1-2p)e^t}]^2 \leq \frac{pe^t}{1-(1-p)e^t}$$

Proof. The claim is true iff $4pe^t(1-(1-p)e^t) \leq (1-(1-2p)e^t)^2$ which simplifies to $0 \leq (1-e^t)^2$. ☐

Lemma A.1.2. *If $\delta \in [0,1)$ and $p \in [0,1)$, then there exists $e^t < 1/(1-p)$ satisfying*

$$\frac{pe^t}{[1-(1-p)e^t]e^{(1+\delta)t/p}} \leq e^{-\delta^2/2}$$

Proof. Setting $e^{-t} = (1-p)(1+\delta)/(1+\delta-p)$, the expression above simplifies to $(1+\delta)\left[1-\frac{p\delta}{1+\delta-p}\right]^{\frac{1+\delta-p}{p}}$. Using the fact that $(1-\frac{\delta}{m})^m \leq 1-\delta+\delta^2/2$ for positive $m > \delta$, the expression is no more than $(1+\delta)(1-\delta+\delta^2/2) = 1-\delta^2(1+\delta)/2 \leq e^{-\delta^2(1+\delta)/2} < e^{-\delta^2/2}$. It may be verified that $e^t < 1/(1-p)$ assuming $\delta \geq 0$ and $p \in [0,1)$. □

Lemma A.1.3. *Let $N = 2^\ell$. Let c be a sufficiently large constant that satisfies*

$$(\delta^2/2)2^{\lceil \log_2 \ell \rceil + c - 1} \geq 2 \log N$$

Then

$$\Pr[Y(\ell) > (1+\delta)\mathbf{E}Y(\ell)] < 1/N^2$$

Proof. We employ the well-known Chernoff technique as outlined in Motwani and Raghavan [MR95]. Let $(*)$ denote $\Pr[Y(\ell) > (1+\delta)N]$. Then

$$\begin{aligned}
(*) &= \Pr[\exp(tY(\ell)) > \exp((1+\delta)tN)] \\
&= \Pr[\exp(tY(\ell)) > \exp\left((1+\delta)t\sum_{i=0}^{\ell}\frac{\chi(i)}{p(i)}\right)] \\
&\leq [\mathbf{E}\exp(tY(\ell))] / \exp\left((1+\delta)t\sum_{i=0}^{\ell}\frac{\chi(i)}{p(i)}\right) \qquad \text{(by Markov's Inequality)}
\end{aligned}$$

Now

$$\mathbf{E}\exp(tY(\ell)) = \Pi_{i=0}^{\ell}\Pi_{j=1}^{\chi(i)}\mathbf{E}\exp(ty_{ij})$$

where y_{ij} is a geometric variable with parameter $p(i)$. Now $\mathbf{E}\exp(ty_{ij}) = \frac{p(i)e^t}{1-(1-p(i))e^t}$, provided $(1-p(i))e^t < 1$. Note that this introduces the constraint $e^t < 1/(1-p(i))$.

Therefore,

$$\mathbf{E}\exp(tY(\ell)) = \Pi_{i=0}^{\ell}\left[\frac{p(i)e^t}{1-(1-p(i))e^t}\right]^{\chi(i)}$$

which yields

$$
\begin{aligned}
(*) \quad &\leq \quad \left(\Pi_{i=0}^{\ell}\left[\frac{p(i)e^t}{1-(1-p(i))e^t}\right]^{\chi(i)}\right) / \exp\left((1+\delta)t\sum_{i=0}^{\ell}\frac{\chi(i)}{p(i)}\right) \\
&= \quad \Pi_{i=0}^{\ell}\left[\frac{p(i)e^t}{[1-(1-p(i))e^t]\exp\frac{(1+\delta)t}{p(i)}}\right]^{\chi(i)}
\end{aligned}
$$

By repeated application of Lemma A.1.1, we transform the above expression into

$$
(*) \quad \leq \quad \Pi_{i=0}^{\ell}\left[\frac{p(i)e^t}{[1-(1-p(i))e^t]\exp\frac{(1+\delta)t}{p(i)}}\right]^{\eta(i)}
$$

where $\eta(i) = 0$ or $\eta(i) = 1$ for $0 \leq i \leq \ell - 1$, and $\eta(\ell) \geq \chi(\ell)$.

For $i < \ell$, each of the terms in the product above is no more than 1. Ignoring them, we are left with the expression

$$
(*) \quad \leq \quad \left[\frac{p(\ell)e^t}{[1-(1-p(\ell))e^t]\exp\frac{(1+\delta)t}{p(\ell)}}\right]^{\eta(\ell)}
$$

We now use Lemma A.1.2 to claim that

$$
(*) \quad \leq \quad e^{-\delta^2(1+\delta)\eta(\ell)/2}
$$

Now $\eta(\ell) \geq \chi(\ell)$ where $\chi(\ell) = 2^{\lceil \log_2 \ell \rceil + c - 1}$. Recall that $N = 2^{\ell}$ and that we chose constant c such that $(\delta^2/2)(1+\delta)2^{\lceil \log_2 \ell \rceil + c - 1} \geq 2 \log N$. Therefore,

$$
(*) \quad \leq \quad e^{-2\log N} = 1/N^2
$$

$\square$

Lemma 2.3.1(a) follows directly from Lemma A.1.3.

A.2 Proof of Lemma 2.3.1(b)

Lemma A.2.1. *If $\delta \in [0, \frac{1}{2})$ and $p \in [0, 1)$, then there exists $e^t > 1 - p$ satisfying*

$$\frac{pe^{-t}}{[1-(1-p)e^{-t}]e^{-(1-\delta)t/p}} \le e^{-5\delta^2(1-14\delta/15)/2}$$

Proof. With $e^t = 1 - \frac{p\delta}{1-\delta}$, the expression simplifies to $\frac{1-\delta}{1-2\delta}\left(1 - \frac{p\delta}{1-\delta}\right)^{\frac{1-\delta}{p}}$. Using the inequality $(1-\frac{\delta}{m})^m < 1-\delta+\delta^2/2$ for $m > \delta$, the expression is no more than $1-5\delta^2(1-14\delta/15)/2 < e^{-5\delta^2(1-14\delta/15)/2}$. It may be verified that $e^t > (1-p)$ for $\delta \in [0, \frac{1}{2})$ and $p \in [0, 1)$. □

Lemma A.2.2. *Let $N = 2^\ell$. Let c be a sufficiently large constant that satisfies*

$$(5\delta^2/2)(1 - 14\delta/15)2^{\lceil \log_2 \ell \rceil + c - 1} \ge 2 \log N$$

Then

$$\Pr[Y(\ell) < (1-\delta)\mathbf{E}Y(\ell)] < 1/N^2$$

Proof. Let $(**)$ denote $\Pr[Y(\ell) < (1-\delta)N]$. Then

$$\begin{aligned} (**) &= \Pr[-Y(\ell) > -(1-\delta)N] \\ &= \Pr[\exp(-tY(\ell)) > \exp(-(1-\delta)tN)] \end{aligned}$$

Continuing in the same manner as Lemma A.1.3, we finally obtain

$$(**) \le \left[\frac{p(\ell)e^{-t}}{[1-(1-p(\ell))e^{-t}]\exp\frac{-(1-\delta)t}{p(\ell)}}\right]^{\eta(\ell)}$$

while introducing the constraint $e^{-t}(1 - p(\ell)) < 1$, which is equivalent to $e^t > (1 - p(\ell))$. Using Lemma A.2.1, we obtain

$$(**) \le e^{-5\delta^2(1-15\delta/14)\eta(\ell)/2}$$

Substituting the value of c we chose, we obtain

$$(**) \le e^{-2\log N} = 1/N^2$$

□

Bibliography

[A01] K Aberer. P-Grid: A self-organizing access structure for P2P information systems. *Proc. 9th Intl. Conference on Cooperative Information Systems (CoopIS 2001)*, p. 179 – 194, 2001.

[A02] K Aberer. Scalable data access in peer-to-peer systems using unbalanced search trees. *Proc. 4th International Workshop on Distributed Data and Structures (WDAS 2002)*, p. 107 – 120, 2002.

[A04] N Alon. Problems and results in extremal combinatorics, II. Available as `http://www.math.tau.ac.il/~nogaa/PDFFS/publications.html`, 2004.

[AAA+03] I Abraham, B Awerbuch, Y Azar, Y Bartal, D Malkhi, and E Pavlov. A generic scheme for building overlay networks in adversarial scenarios. *Proc. Intl. Parallel and Distributed Processing Symposium (IPDPS 2003)*, 2003.

[ABKU99] Y Azar, A Z Broder, A R Karlin, and E Upfal. Balanced allocations. *SIAM Journal of Computing*, 29(1):180–200, 1999.

[ACLW02] D Anderson, J Cobb, E K M Lebofsky, and D Werthimer. SETI@home: An experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.

[ACMD+03] K Aberer, P Cudré-Mauroux, A Datta, Z Despotovic, M Hauswirth, M Punceva, and R Schmidt. P-Grid: A self-organizing structured P2P system. *SIGMOD Record*, 32(2), 2003.

[ADS02] J Aspnes, Z Diamadi, and G Shah. Fault-tolerant routing in peer-to-peer systems. *Proc. 21st ACM Symposium on Principles of Distributed Computing (PODC 2002)*, p. 223–232, 2002.

[AHKV03] M Adler, E Halperin, R M Karp, and V V Vazirani. A stochastic process on the hypercube with applications to peer-to-peer networks. *Proc. 35nd ACM Symposium on Theory of Computing (STOC 2003)*, p. 575–584, 2003.

[AKK04] J Aspnes, J Kirsch, and A Krishnamurthy. Load balancing and locality in range-queriable data structures. *Proc. 23rd ACM Symposium on Principles of Distributed Computing (PODC 2004)*, 2004.

[AMD04] I Abraham, D Malkhi, and O Dobzinski. LAND: Stretch $(1+\epsilon)$ locality-aware networks for DHTs. *Proc. 15th ACM-SIAM Symposium on Discrete Algorithms (SODA 2004)*, p. 543–552, 2004.

[AMM04] I Abraham, D Malkhi, and G S Manku. The degree-diameter greedy routing problem. *Manuscript*, 2004.

[AN86] M Aizenman and C M Newman. Discontinuity of the percolation density in one dimensional $1/|x-y|^2$ percolation models. *Communications in Mathematical Physics*, 107:611–647, 1986.

[AS03] J Aspnes and G Shah. Skip graphs. *Proc. 14th ACM-SIAM Symposium on Discrete Algorithms (SODA 2003)*, p. 384–393, 2003.

[B01] B Bollobás. *Random Graphs*. Cambridge University Press, 2nd edition, 2001.

[B02] A L Barabási. *Linked: The New Science of Networks*. Perseus Publishing, 2002.

[BB01] I Benjamini and N Berger. The diameter of a longe-range percolation clusters on finite cycles. *Random Structures and Algorithms*, 19(2):102–111, 2001.

[BC88] B Bollobás and F R K Chung. The diameter of a cycle plus a random matching. *SIAM Journal on Discrete Mathematics*, 1(3):328–333, 1988.

[BCM04] J W Byers, J Considine, and M Mitzenmacher. Geometric generalizations of the power of two choices. *Proc. 16th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2004)*, p. 54–63, 2004.

[BDET00] W J Bolosky, J R Douceur, D Ely, and M Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. *ACM SIGMETRICS 2000*, p. 34–43, 2000.

[BDQ92] J C Bermond, C Delorme, and J J Quisquater. Table of large (δ, d)-graphs. *Discrete Applied Mathematics*, 37/38:575–577, 1992.

[BFKK01] L Barrière, P Fraigniaud, E Kranakis, and D Krizanc. Efficient routing in networks with long range contacts. *Proc. 15th Intl. Symposium on Distributed Computing (DISC 2001)*, p. 270–284, 2001.

[BLNS82] A D Birrell, R Levin, R M Needham, and M D Shroeder. Grapevine: An exercise in distributed computing. *Communications of the ACM*, 25(4):260–274, 1982.

[BM72] R Bayer and E M McCreight. Organization and maintenance of large ordered indexes. *Acta Informatica*, 1:173–189, 1972.

[BMR03] M Bawa, G S Manku, and P Raghavan. SETS: Search Enhanced by Topic Segmentation. *Proc. 26th Annual ACM SIGIR Conference (SIGIR 2003)*, p. 306–313, 2003.

[C52] H Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathmematical Statistics*, 23(4):493–509, 1952.

[CCR+03] B Chun, D Culler, T Roscoe, A Bavier, L Peterson, M Wawrzoniak, and M Bowman. Planetlab: An overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communications Review*, 33(3):3–12, 2003.

[CDHR03] M Castro, P Druschel, Y C Hu, and A I T Rowstron. Topology-aware routing in structured peer-to-peer overlay networks. *Proc. Intl. Workshop on Future Directions in Distrib. Computing (FuDiCo 2003)*, p. 103–107, 2003.

[CG92] F Comellas and J Gómez. New large graphs with given degree and diameter. *Graph Theory, Combinatorics and Algorithms*, 1:221–233, 1992.

[CGH+04] G Cordasco, L Gargano, M Hammar, A Negro, and V Scarano. F-Chord: Improved uniform routing on Chord. *Proc. 11th Colloquium on Structural Information and Communication Complexity*, 2004.

[CGS02] D Coppersmith, D Gamarnik, and M Sviridenko. The diameter of a longe-range percolation graph. *Random Structures and Algorithms*, 21(1):1–13, 2002.

[CHM+02] I Clarke, T Hong, S Miller, O Sandberg, and B Wiley. Protecting free expression online with Freenet. *IEEE Internet Computing*, 6(1):40–49, 2002.

[CRB+03] Y Chawathe, S Ratnasamy, L Breslau, N Lanham, and S Shenker. Making Gnutella-like P2P systems scalable. *Proc. ACM SIGCOMM 2003*, 2003.

[D04] C Delorme. The (Degree, Diameter) problem for graphs. Laboratoire de Recherche en Informatique, Université Paris Sud, France. Available as `http://maite71.upc.es/grup\_de\_grafs/table\_g.html`, 2004.

[dB46] N G d Bruijn. A combinatorial problem. *Proc. Koninklijke Nederlandse Akademie van Wetenschappen*, 49:758–764, 1946.

[DCD+04] F Dabek, R Cox, F Dabek, F Kaashoek, and R Morris. Vivaldi: A decentralized network coordinate system. *Proc. ACM SIGCOMM 2004*, 2004.

[DKK+01] F Dabek, M F Kaashoek, D Karger, R Morris, and I Stoica. Wide-area cooperative storage with CFS. *Proc. 18th ACM Symposium on Operating Systems Principles (SOSP 2001)*, p. 202–215, 2001.

[DLS+04] F Dabek, J Li, E Sit, J Robertson, M F Kaashoek, and R Morris. Designing a DHT for low latency and high throughput. *Proc. 1st Symposium on Networked Systems Design and Implementation (NSDI 2004)*, p. 85–98, 2004.

[DMW03] P S Dodds, R Muhamad, and D J Watts. An experimental study of search in global social networks. *Science*, 301:827–829, 2003.

[DR98] D P Dubhashi and D Ranjan. Balls and bins: A study in negative dependence. *Random Structures and Algorithms*, 13(2):99–124, 1998.

[DYN03] J Duato, S Yalamanchili, and L M Ni. *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann, 2 edition, 2003.

[E01a] D Eastlake. US Secure Hash Algorithm 1 (sha1). RFC 3174, IETF, 2001.

[E01b] G Exoo. A family of graphs and the degree/diameter problem. *J. of Graph Theory*, 37:118–124, 2001.

[FG03] P Fraigniaud and P Gauron. (brief announcement) An overview of the content-addressable network D2B. *Proc 22nd ACM Symposium on Principles of Distributed Computing (PODC 2003)*, p. 151–151, 2003.

[FGP04] P Fraigniaud, C Gavoille, and C Paul. Eclecticism shrinks even small worlds. *Proc 23rd ACM Symposium on Principles of Distributed Computing (PODC 2004)*, p. 169–178, 2004.

[G81] G H Gonnet. Expected length of the longest probe sequence in hash code searching. *Journal of the ACM*, 28(2):289–304, 1981.

[GBGM04] P Ganesan, M Bawa, and H Garcia-Molina. Online balancing of range-partitioned data with applications to peer-to-peer systems. *Proc. 30th Intl. Conf. on Very Large Data Bases (VLDB 2004)*, 2004.

[GBHC00] S D Gribble, E A Brewer, J M Hellerstein, and D Culler. Scalable, distributed data structures for Internet service construction. *Proc. 4th Symposium on Operating System Design and Implementation (OSDI 2000)*, p. 319–332, 2000.

[GDS+03] K P Gummadi, R J Dunn, S Saroiu, S D Gribble, H M Levy, and J Zahorjan. Measurement, modeling and analysis of a peer-to-peer file-sharing workload. *Proc. 19th ACM Symposium on Operating Systems Principles (SOSP 2003)*, 2003.

[GGG+03] K P Gummadi, R Gummadi, S D Gribble, S Ratnasamy, S Shenker, and I Stoica. The impact of DHT routing geometry on resilience and proximity. *Proc. ACM SIGCOMM 2003*, p. 381–394, 2003.

[GLS+04] B Godfrey, K Lakshminarayanan, S Surana, R M Karp, and I Stoica. Load balancing in dynamic structured P2P systems. *Proc. IEEE INFOCOM 2004*, 2004.

[GM04] P Ganesan and G S Manku. Optimal routing in Chord. *Proc. 15th ACM-SIAM Symposium on Discrete Algorithms (SODA 2004)*, p. 169–178, 2004.

[GS04] P B Godfrey and I Stoica. Heterogeneity and load balance in distributed hash tables. Technical report, University of California at Berkeley, 2004.

[H63] W Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[HJS+03] N J A Harvey, M Jones, S Saroiu, M Theimer, and A Wolman. SkipNet: A scalable overlay network with practical locality properties. *Proc. 4th USENIX Symposium on Internet Technologies and Systems (USITS 2003)*, 2003.

[HKMR04] K Hildrum, J Kubiatowicz, S Ma, and S Rao. A note on the nearest neighbour in growth-restricted metrics. *Proc. 15th ACM-SIAM Symposium on Discrete Algorithms (SODA 2004)*, p. 553–554, 2004.

[HM03a] N Harvey and J I Munro. (brief announcement) Deterministic SkipNet. *Proc 22nd ACM Symposium on Principles of Distributed Computing (PODC 2003)*, p. 152–153, 2003.

[HM03b] K Horowitz and D Malkhi. Estimating network size from local information. *Information Processing Letters*, 88(5):237–243, 2003.

[HMU00] J E Hopcroft, R Motwani, and J D Ullman. *Introduction to Automata Theory, Languages and Computation.* Pearson Addison Wesley, 2 edition, 2000.

[IRD02] S Iyer, A I T Rowstron, and P Druschel. Squirrel: A decentralized, peer-to-peer web cache. *Proc. 21st ACM Symposium on Principles of Distributed Computing (PODC 2002)*, p. 213–222, 2002.

[JK77] N L Johnson and S Kotz. *Urn Models and their Applications: An Approach to Modern Discrete Probability Theory.* John Wiley and Sons, 1977.

[K68] W H Kautz. Bounds on directed (d, k) graphs. *Theory of Cellular Logic Networks and Machines (AFCRL-68-0668, SRI Project 7258, Final Report)*, p. 20–28, 1968.

[K69] W H Kautz. Design of optimal interconnection networks for multiprocessors. *Architecture and Design of Digital Computers (Nato Advanced Summer Institute)*, p. 249–272, 1969.

[K00] J Kleinberg. The small-world phenomenon: An algorithmic perspective. *Proc. 32nd ACM Symposium on Theory of Computing (STOC 2000)*, p. 163–170, 2000.

[KBC+00] J Kubiatowicz, D Bindel, Y Chen, P Eaton, D Geels, R Gummadi, S Rhea, H Weatherspoon, W Weimer, C Wells, and B Y Zhao. Oceanstore: An architecture for global-scale persistent storage. *Proc. 9th Intl. conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, p. 190–201, 2000.

[KK03] M F Kaashoek and D R Karger. Koorde: A simple degree-optimal hash table. *Proc. 2nd Intl. Workshop on Peer-to-Peer Systems (IPTPS 2003)*, p. 98–107, 2003.

[KM04] K Kenthapadi and G S Manku. Structured coupon collection over cliques for P2P load balancing. Technical Report DB Group TR 2004-38, Computer Science Department, Stanford University, CA, USA, 2004.

[KMXY03] A Kumar, S Merugu, J J Xu, and X Yu. Ulysses: A robust, low-diameter, low-latency peer-to-peer network. *Proc. 11th IEEE International Conference on Network Protocols (ICNP 2003)*, 2003.

[KR02] D R Karger and M Ruhl. Finding nearest neighbors in growth-restricted metrics. *Proc. 34th ACM Symposium on Theory of Computing (STOC 2002)*, p. 741–750, 2002.

[KR04] D R Karger and M Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. *Proc. 16th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2004)*, p. 36–43, 2004.

[KS04] V King and J Saia. Choosing a random peer. *Proc. 23rd ACM Symposium on Principles of Distributed Computing (PODC 2004)*, p. 125–130, 2004.

[KSC78] V F Kolchin, B A Sevast'yanov, and V P Chistyakov. *Random Allocations.* V H Winston & Sons, 1978.

[L83] B W Lampson. Hints for computer system design. *ACM Operating Systems Review*, 15(5):33–48, 1983.

[L92] F T Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays - Trees - Hypercubes.* Academic Press/Morgan Kaufmann, 1992.

[LKRG03] D Loguinov, A Kumar, V Rai, and S Ganesh. Graph-theoretic analysis of structured peer-to-peer systems: Routing distance and fault resilience. *Proc. ACM SIGCOMM 2003*, p. 395–406, 2003.

[LMP04] X Li, J Misra, and G Plaxton. Active and concurrrent topology maintenance. *Proc. 18th Intl. Symposium on Distributed Computing (DISC 2004)*, 2004.

[LNBK02] D Liben-Nowell, H Balakrishnan, and D R Karger. Analysis of the evolution of peer-to-peer systems. *Proc. 22nd ACM Symposium on Principles of Distriuted Systems (PODC 2002)*, p. 233–242, 2002.

[LNS96] W Litwin, M A Neimat, and D A Schneider. LH* – A scalable, distributed data structure. *ACM Transactions on Database Systems*, 21(4):480–525, 1996.

[LS04] E Lebhar and N Schabanel. Close to optimal decentralized routing in long-range contact networks. *Proc. 31st Intl. Colloq. on Automata, Languages and Programming (ICALP 2004)*, 2004.

[M67] S Milgram. The small world problem. *Psychology Today*, 67(1):60–67, 1967.

[M96] M Mitzenmacher. *The Power of Two Choices in Randomized Load Balancing*. PhD dissertation, University of California at Berkeley, Department of Computer Science, 1996.

[M03] G S Manku. Routing networks for distributed hash tables. *Proc. 22nd ACM Symposium on Principles of Distributed Computing (PODC 2003)*, p. 133–142, 2003.

[M04] G S Manku. Balanced binary trees for ID management and load balance in distributed hash tables. *Proc. 23rd ACM Symposium on Principles of Distributed Computing (PODC 2004)*, p. 197–205, 2004.

[MBR03] G S Manku, M Bawa, and P Raghavan. Symphony: Distributed hashing in a small world. *Proc. 4th USENIX Symposium on Internet Technologies and Systems (USITS 2003)*, p. 127–140, 2003.

[MD88] P V Mockapetris and K J Dunlap. Developnment of the Domain Name System. *Proc. ACM SIGCOMM 1988*, p. 123–133, 1988.

[MN04] C Martel and V Nguyen. Analyzing kleinberg's (and other) small-world models. *Proc 23rd ACM Symposium on Principles of Distributed Computing (PODC 2004)*, p. 179–188, 2004.

[MNR02] D Malkhi, M Naor, and D Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. *Proc 21st ACM Symposium on Principles of Distributed Computing (PODC 2002)*, p. 183–192, 2002.

[MNW04] G S Manku, M Naor, and U Wieder. Know thy neighbor's neighbor: The power of lookahead in randomized P2P networks. *Proc. 36th ACM Symposium on Theory of Computing (STOC 2004)*, p. 54–63, 2004.

[MR95] R Motwani and P Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[MRS01] M Mitzenmacher, A W Richa, and R Sitaraman. The power of two random choices: A survey of techniques and results. *Handbook of Randomized Computing (Vol 1).* Kluwer Academic Press, 2001.

[NS86] C M Newman and L S Schulman. One dimensional $1/|j-i|^s$ percolation models: The existence of a transition for $s \leq 2$. *Communications in Mathematical Physics*, 180:483–504, 1986.

[NW03] M Naor and U Wieder. Novel architectures for P2P applications: The continuous-discrete approach. *Proc. 15th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2003)*, p. 50–59, 2003.

[NWS02] M E J Newman, D J Watts, and S H Strogatz. Random graph models of social networks. *Proc. National Academy of Science, USA*, 99 (suppl 1):2566–2572, 2002.

[NZ02] T S E Ng and H Zhang. Predicting Internet network distance with coordinate-based approaches. *Proc. IEEE INFOCOM 2002*, 2002.

[P90] W Pugh. Skip lists: A probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, 1990.

[PK78] I Pool and M Kochen. Contacts and influence. *Social Networks*, 1:1–48, 1978.

[PRR99] C G Plaxton, R Rajaraman, and A W Richa. Accessing nearby copies of replicated objects in a distributed environment. *Theory of Computing Systems*, 32(3):241–280, 1999.

[R92] R L Rivest. The MD5 Message-Digest Algorithm. RFC 1321, IETF, 1992.

[RB03] M Roussopoulos and M Baker. CUP: Controlled Update Propagation in peer-to-peer networks. *Proc. 2003 USENIX Annual Technical Conference*, p. 167–180, 2003.

[RD01a] A I T Rowstron and P Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, p. 329–350, 2001.

[RD01b] A I T Rowstron and P Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. *Proc 18th ACM Symposium on Operating Systems Principles (SOSP 2001)*, p. 188–201, 2001.

[RFHK01] S Ratnasamy, P Francis, M Handley, and R M Karp. A scalable Content-Addressable Network. *Proc. ACM SIGCOMM 2001*, p. 161–172, 2001.

[RFI02] M Ripeanu, I Foster, and A Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *Internet Computing Journal*, 6(1), 2002.

[RGRK04] S Rhea, D Geels, T Roscoe, and J Kubiatowicz. Handling churn in a DHT. *Proc. 2004 USENIX Annual Technical Conference*, 2004.

[RHKS01] S Ratnasamy, M Handley, R M Karp, and S Shenker. Application-level multicast using content addressable networks. *Proc. 3rd Intl. Networked Group Communication Workshop (NGC 2001)*, p. 14–29, 2001.

[RKCD01] A I T Rowstron, A M Kermarrec, M Castro, and P Druschel. SCRIBE: The design of a large-scale event notification infrastructure. *Proc. 3rd Intl. Networked Group Communication Workshop (NGC 2001)*, p. 30–43, 2001.

[RS98] M Raab and A Steger. Balls into bins – a simple and tight analysis. *Randomization and Approximation Techniques in Computer Science (RANDOM 1998), Lecture Notes in Computer Science 1518*, p. 159–170, 1998.

[RS04a] V Ramasubramanian and E G Sirer. Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays. *Proc. 1st Symposium on Networked Systems Design and Implementation (NSDI 2004)*, p. 99–112, 2004.

[RS04b] V Ramasubramanian and E G Sirer. The design and implementation of a next generation name service for the Internet. *Proc. ACM SIGCOMM 2004*, 2004.

[RSS02] S Ratnasamy, S Shenker, and I Stoica. Routing algorithms for DHTs: Some open questions. *Proc. 1st Intl. Workshop on Peer-to-Peer Systems (IPTPS 2002)*, p. 45–52, 2002.

[S83] L S Schulman. Long range percolation in one dimension. *Journal of Physics A*, 16(17):L639–L641, 1983.

[S01] J H Spencer. *The Strange Logic of Random Graphs*. Springer Verlag, 2001.

[SGG02] S Saroiu, P K Gummadi, and S D Gribble. A measurement study of peer-to-peer file sharing systems. *Proceedings of the Multimedia Computing and Networking (MMCN'02)*, 2002.

[Ski] The Skitter Project. http://www.caida.org.

[SMK+01] I Stoica, R Morris, D Karger, M F Kaashoek, and H Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. *Proc. ACM SIGCOMM 2001*, p. 149–160, 2001.

[V97] S Vinoski. CORBA: Integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine*, 14(2), 1997.

[WK02] H Weatherspoon and J D Kubiatowicz. Erasure coding vs replication: A quantitative comparison. *Proc. 1st Intl. Workshop on Peer-to-Peer Systems (IPTPS 2002)*, 2002.

[WS98] D Watts and S Strogatz. Collective dynamics of small-world networks. *Nature*, p. 440–442, 1998.

[XKY03] J Xu, A Kumar, and X Yu. On the fundamental tradeoff between routing table size and network diameter in peer-to-peer networks. *Proc. IEEE INFOCOM 2003*, 2003.

[YGM01] B Yang and H Garcia-Molina. Comparing hybrid peer-to-peer systems. *Proc. 27th Intl. Conf. on Very Large Data Bases (VLDB 2001)*, 2001.

[ZCB96] E W Zegura, K L Calvert, and S Bhattacharjee. How to model an Internetwork. *Proc. IEEE INFOCOM 1996*, 1996.

[ZGG03] H Zhang, A Goel, and R Govindan. Incrementally improving lookup latency in distributed hash table systems. *ACM SIGMETRICS 2003*, p. 114–125, 2003.

[ZHS+04] B Y Zhao, L Huang, J Stribling, S C Rhea, A D Joseph, and J D Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1), 2004.

[ZZJ+01] S Zhuang, B Y Zhao, A D Joseph, R H Katz, and J D Kubiatowicz. Bayeux: An architecture for wide-area, fault-tolerant data dissemination. *Proc. 11th Intl. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 2001)*, p. 11–20, 2001.