

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

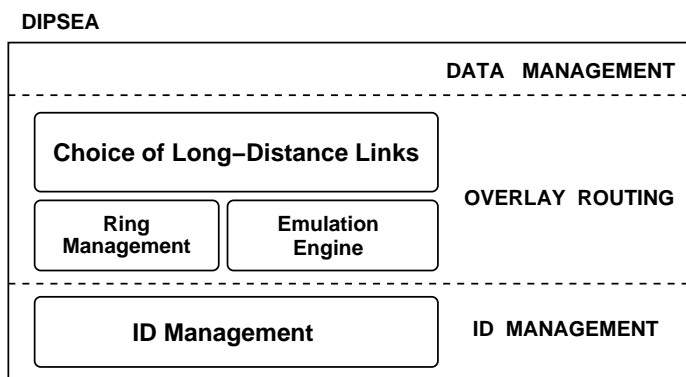


Figure 1.1: Dipsea: A three-layered architecture for building Distributed Hash Tables. Efficient algorithms for ID Management are described in Chapters 2 and 3. See Section 1.3 for a brief description of Ring Management. See Chapter 4 for the Emulation Engine, and Chapters 5 through 9 for Choice of Long-Distance 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: $\lceil \log_2 n \rceil$ and $\lceil \log_2 n \rceil \pm 1$, where $\lceil x \rceil$ 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 $\langle G_0, G_1, G_2, \dots \rangle$ 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 $\mathbf{x}_1^1, \mathbf{x}_1^2, \dots, \mathbf{x}_1^{i_1}$ denote the i_1 neighbors of label \mathbf{x}_1 in graph $G_{\ell-2}$. Let $\mathbf{x}_2^1, \mathbf{x}_2^2, \dots, \mathbf{x}_2^{i_2}$ denote the i_2 neighbors of label \mathbf{x}_2 in graph $G_{\ell-3}$. Let $\mathbf{x}_3^1, \mathbf{x}_3^2, \dots, \mathbf{x}_3^{i_3}$ 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: $C(\mathbf{x}_1^1), C(\mathbf{x}_1^2), \dots, C(\mathbf{x}_1^{i_1}), C(\mathbf{x}_2^1), C(\mathbf{x}_2^2), \dots, C(\mathbf{x}_2^{i_2})$, and $C(\mathbf{x}_3^1), C(\mathbf{x}_3^2), \dots, C(\mathbf{x}_3^{i_3})$. 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:

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

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 $\lceil \log_2 n \rceil$ or $\lceil \log_2 n \rceil \pm 1$, where n is the total number of leaf nodes, and $\lceil x \rceil$ denotes the integer closest to real number x . Therefore, each cluster on $\lceil \log_2 n \rceil - 1$ 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 $\lceil \log_2 n \rceil - 1$ is a member of all the triplets. Therefore, a message will eventually be delivered to a cluster on $\lceil \log_2 n \rceil - 1$ 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 $1+d+d^2+\dots+d^\Delta = \frac{d^{\Delta+1}-1}{d-1}$, 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 $m = O(\text{poly}(k))$, 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 $[\mathbf{x} + r] \bmod n$.

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 $\delta : V \times V \rightarrow \mathcal{R}^+$, 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 $1 \leq i \leq \ell$, 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 $0 \leq i < \ell$, 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 $k \geq 1$ *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 $[\mathbf{x} + r] \bmod n$. 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: $\langle \frac{n}{2}, \frac{n}{4}, \frac{n}{8}, \dots, 4, 2 \rangle$. 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 [ACW02], 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 [KMX03]. 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.