The Continuous-Discrete Approach for Designing P2P Networks and Algorithms

Thesis for the Ph.D. Degree

by

Udi Wieder

Under the Supervision of
Professor Moni Naor

Department of Computer Science and Applied Mathematics
The Weizmann Institute of Science

Submitted to the Feinberg Graduate School of
the Weizmann Institute of Science
Rehovot 76100, Israel

August 10, 2005

To my parents Freddy and Hana Wieder

Abstract

One of the most intriguing trends in the development of internet applications in past few years is the immense rise in popularity of *peer-to-peer* (P2P) applications and networks. Peer-to-peer networks are characterized by the lack of central control or a-priori hierarchical organization in which all or most communication is symmetric. Moreover a P2P system is expected to scale gracefully as the size of the network grows. The sheer scale and dynamism in which P2P networks are suppose to operate make the design of P2P systems challenging even for relatively simple applications.

The thesis is divided into two main parts. In the first we propose a new general approach for designing P2P networks called *the continuous discrete* approach. We demonstrate our technique by suggesting several specific distributed data structures. The first is a design for the distributed hash table (DHT) abstract data structure which we call `Distance Halving` (DH). Our design has an optimal tradeoff between node degree and path hop length. A variant of our design is also robust against random faults and even against a form of adversarial behavior of a random subset of the nodes (which we call the spam generating model). An important feature of the DH hash table is a dynamic caching algorithm which completely prevents hot spots.

The second data structure we suggest is a dynamic quorum system called *Dynamic Paths*. We prove some new lower bounds on the probe complexity of quorum systems. We then show that *Dynamic Paths* is optimal with respect to load, availability and probe complexity.

Finally we show how the continuous-discrete approach could be used to emulate in a P2P setting *any* topology, thus basically showing that all the classical inter connection topologies (studied in the context parallel computing) may serve as a basis for a P2P network.

In the second part of the Thesis we analyze some new algorithms for known P2P networks. Our main result shows that routing algorithms which take into account neighbors of neighbors for performing better routing decisions, are often provably better than Greedy algorithms and in some cases optimal. We analyze the Neighbor of Neighbor algorithms in the context of small world networks and Skip-Graphs. Finally we show that Skip-Graphs are in fact expanders with high probably and show how to utilize this fact for a variety of algorithms.

Acknowledgments

I am in great debt to my advisor Moni Naor. Moni treated me as a friend and a peer from the first moment, building within me both confidence and motivation. Without his guidance and strong feedback this thesis would not have materialized. Moni is not only a brilliant researcher but also an intellectual in the full sense of the word. One can learn something new from every conversation with Moni, however short (and it rarely is). I hope and look forward to continued collaboration with him in the future.

Like many students before me, I feel privileged to have been given the opportunity to study and research in the Weizmann Institute. The atmosphere in our department wonderfully combines academic excellence with openness and friendliness. There is no doubt in my mind that this atmosphere had been a key ingredient in the successful completion of my PhD. Throughout my studies I learned much from the faculty members of the department and from fellow students. I would like to especially mention Uri Feige who has been my advisor during my studies towards an M.Sc degree. From Uri I first learned how to become a researcher, he still serves as a role model for me and as a source of inspiration. Special thanks to Itai Benjamini for being my friend and collaborator and for introducing to me many mathematical ideas which found their way into my research papers.

The years in Weizmann would not have been the same without my friends. I thank my seniors Michael Langberg, Yehuda Lindell, Omer Reingold, Alon Rosen, Ronen Shaltiel and Amir Shpilka for helping me find my grounds and sharing with me their knowledge. I am very grateful to my mates Boaz Barak, Danny Harnik, Yiftach Heitner, Eran Tromer with which I had the pleasure to work and play. Special thanks to Eran Ofek who endured me as a friend and an office-mate for six straight years.

Last but most important, I'd like to express my deepest gratitude to my family for their constant love and support. My parents Freddy and Hana who implanted within me curiosity and desire for knowledge. Special thanks to my wife Yael, without whose patience and encouragement I probably would not have managed to bring my studies to completion. My sweet daughter Noa played a special role by constantly reminding me what's *really* important.

Contents

Abstract . . . iii
Acknowledgments . . . v

1 Introduction **1**
1.1 The P2P model . . . 1
1.1.1 Measures of Quality . . . 2
1.1.2 The Continuous-Discrete Approach for Designing P2P Networks . . . 2
1.2 Thesis Organization . . . 3

2 Distributed Hash Tables **5**
2.1 Introduction . . . 5
2.1.1 Our Contributions . . . 7
2.2 The Distance Halving DHT . . . 8
2.2.1 The Construction . . . 8
2.2.2 Joining and Leaving the Network . . . 10
2.2.3 The Lookup Operation . . . 10
2.2.4 Permutation Routing . . . 14
2.2.5 Degree-Path Length Optimality . . . 16
2.3 Fault Tolerant constructions . . . 17
2.3.1 Related Work . . . 17
2.3.2 The Overlapping Distance Halving DHT . . . 17
2.3.3 The Lookup Operation . . . 19
2.4 Dynamic Caching - Relieving Hot Spots . . . 21
2.4.1 The Protocol . . . 23
2.4.2 The Active Tree . . . 24
2.4.3 Analysis of a Single Hotspot . . . 25
2.4.4 Multiple Hotspots . . . 27
2.5 Emulating General Graphs - Smoothness is Everything . . . 28
2.6 Load Balancing the ID's - Achieving Smoothness . . . 29
2.6.1 Handling Deletions . . . 31
2.7 Higher Dimensions . . . 33
2.7.1 Dynamic Voronoi Diagrams . . . 33
2.7.2 Achieving Smoothness in Two Dimensions . . . 36
2.7.3 Constructing Expanders . . . 37

3 Scalable and Dynamic Quorum Systems **39**
3.1 Introduction and Motivation . . . 39
3.1.1 Measures of Quality . . . 40
3.1.2 Our Contributions . . . 42
3.2 Non Adaptive Algorithms vs. Load . . . 42
3.3 The Paths Quorum System . . . 44
3.3.1 Algorithmic Probe Complexity in *Paths* . . . 44
3.4 The *Dynamic Paths* Quorum System . . . 50
3.4.1 The Quorum System . . . 51
3.4.2 A Smooth Voronoi Diagram . . . 52
3.4.3 A simpler Quorum System . . . 55
3.5 Conclusion and Open Questions . . . 55

4 The Neighbor of Neighbor Algorithm **57**
4.1 Small World Networks . . . 57
4.1.1 Our Contributions . . . 59
4.1.2 Related Work . . . 59
4.1.3 The NoN-GREEDY Routing Algorithm . . . 60
4.2 NoN in Small-World Percolation Graphs . . . 61
4.3 NoN in Small-World P2P Networks . . . 64
4.4 NoN in Skip Graphs . . . 66
4.4.1 A Review of Skip Graphs . . . 66
4.4.2 Routing in Skip Graphs . . . 67
4.5 Lower Bounds . . . 73
4.5.1 A Probing Model . . . 73
4.5.2 Lower Bounds in the Probing Model . . . 74
4.6 Implementations of NoN . . . 76
4.6.1 A different Implementation . . . 78
4.6.2 System Issues with NoN-GREEDY . . . 78
4.6.3 Fault Tolerance . . . 80
4.7 Discussion . . . 81

5 The Expansion and Mixing Time of Skip Graphs **83**
5.1 Introduction . . . 83
5.1.1 Comparison with previous work . . . 84
5.2 Main result . . . 85
5.2.1 The expansion of the cycle S_ϵ . . . 85
5.2.2 The Buckets . . . 86
5.2.3 Identifying the bucket edges: . . . 89
5.3 How to sample a random node . . . 89
5.3.1 Speeding up the mixing time . . . 90
5.3.2 Empirical evidence . . . 91
5.3.3 The maximal edge heuristic . . . 93
5.4 Applications . . . 94
5.4.1 Fault tolerance . . . 95
5.4.2 Hitting large sets . . . 95

5.4.3 The skip graph as a peer-to-peer data storage system 96

6 Open Problems and Further Research **99**
6.1 Problems Related to Load Balancing 99
6.2 Problems related to Security 100

Bibliography **103**

List of Tables

2.1 Comparison of *expected* performance measures of lookup schemes. 6

5.1 Quality of mixing when $n = 2^{18}$ and buckets are of size at least 4. 92
5.2 Quality of mixing when the walk starts from a random prefix. 93
5.3 The variation distance between the random walk distribution and the stationary distribution, when $n = 2^{18}$,and the maximum edge is used. 95

List of Figures

2.1 The continuous distance halving graph . . . 8
2.2 The spam resistance lookup protocol . . . 20
2.3 The first 2 layers of the path tree. . . . 23
2.4 A mapping of the active tree into nodes. . . . 26
2.5 The cyclic load balancing scheme . . . 33
2.6 Addition of a new generator to a Voronoi diagram. . . . 34
2.7 A smooth Voronoi diagram . . . 35

3.1 The *Paths* quorum system . . . 44
3.2 A possible quorum in *Paths* after failures. . . . 46
3.3 A possible quorum found by the adaptive algorithm . . . 47
3.4 The line graph of a 4×4 grid and its dual. . . . 49
3.5 A possible sample from the worst case distribution of failed edges. . . . 50
3.6 Maintaining the integrity of the quorum system. . . . 52
3.7 The grid G is put on top of the diagram T. . . . 54
3.8 An example of a quorum in *Dynamic Paths*. . . . 55

4.1 The NoN-GREEDY Algorithm. . . . 60
4.2 Different balls in the grid . . . 63
4.3 An example of a skip graph. . . . 67
4.4 Average length of paths in skip graphs . . . 77
4.5 Average length of hops in Small World Percolation graphs . . . 77
4.6 Degree/path length tradeoff in Small World Percolation graphs and skip graphs . . . 78
4.7 Comparison between the two variants of NoN . . . 79
4.8 Optimistic fault tolerance in Skip Graphs . . . 80
4.9 Pessimistic Fault Tolerance in Percolation Small World graphs . . . 80

5.1 The spectral gap of the expanding sub graph . . . 92
5.2 The distribution of the degrees when the maximum edge is taken. . . . 94
5.3 The spectral gap when using the maximum edge heuristic . . . 95

Chapter 1

Introduction

Summary: In this Chapter we describe the P2P model. We list the unique requirements from a good P2P system and illustrate the main challenges facing designers. We sketch the Continuous-Discrete approach which is central to many of the results in the Thesis.

1.1 The P2P model

One of the most intriguing trends in the development of internet applications in past few years is the immense rise in popularity of *peer-to-peer* (P2P) applications and networks. Peer-to-peer networks are characterized by the lack of central control or a-priori hierarchical organization in which all or most communication is symmetric. Moreover a P2P system is expected to scale gracefully as the size of the network grows. The sheer scale and dynamism in which P2P networks are suppose to operate make the design of P2P systems challenging even for relatively simple applications. In the context of the Internet the common model used for designing P2P networks is that of an *overlay network*. In an overlay network all servers are connected via some underlying network (say the IP network). Two servers may open a direct connection (say a TCP channel) if one knows the address of the other. The goal of the network designer is to maintain network structure and functionality while nodes leave and join the network dynamically.

Scalable networks have attracted a considerable amount of attention both in the academic world and in the 'real world' of commercial applications. The first popular application which may be called P2P is the Napster file sharing service. Napster used a traditional client-server design in order to help users locate other users which hold the desired file, then both users exchange the file directly without the intervention of the server. The vital role of a central service implies that Napster does not entirely comply with our P2P definition. Indeed the service was closed when an RIAA lawsuit shut down the server. Subsequently other more decentralized file sharing services emerged. Currently the most popular one is probably eMule [39]. While the motivation for designing P2P file sharing services is legally controversial, other applications such as file distribution (such as bittorrent [23]) and voice over IP (such as Skype [119]) use the P2P paradigm in order to reduce costs and improve bandwidth utilization and thus enjoy better performance over a classical client-server design.

1.1.1 Measures of Quality

The quality of every distributed system (or algorithm) is determined by a number of measures such as the complexity and simplicity of the algorithms involved, the robustness in face of failures and so on. The dynamic nature of P2P-networks imposes some unique requirements:

Cost of join/leave: The network should accommodate changes easily. When nodes join or leave, only a small number of nodes should change their state. In particular the linkage; i.e. the degree of the graph that represents the network, should be small.

Decentralization: The algorithmic task of the network should be distributed among all participants as evenly as possible. This means that no node (or cluster of nodes) acts as a central server.

Load Balancing: Complementing the decentralization requirement, it is also required that the total amount of work is divided among the nodes according to their relative resources. In particular, if all nodes have more or less the same amount of bandwidth and computational power (a common assumption) then the work load should be evenly *balanced* between the nodes.

Scalability: A P2P network is supposed to operate properly and efficiently even when the network size varies from a few dozen nodes to a few millions. This means that load imposed on each node should grow very slowly with the size of the network.

Locality: The construction of the P2P overlay network should be sensitive to the metric hidden by the underlying network (say the IP network). It is desirable that operations would not be efficient only in terms of the overlay network but would also take into account the varying latencies of the channels.

1.1.2 The Continuous-Discrete Approach for Designing P2P Networks

We present a high-level description of the framework which may be titled "think continuously, act discretely". Let I be some continuous set in an Euclidean space (typically $I = [0, 1)$ but higher dimensions are also useful). Let G_c be a graph where the vertex set is I. Each point in I is connected to some other points. The actual network is a *discretization* of this continuous graph based on a dynamic decomposition of the underlying space I into cells where each node is responsible for a cell. Let $C_1, C_2 \subseteq I$ be two cells. The node responsible for C_1 is connected to the node responsible to C_2 iff there exists $x \in C_1, y \in C_2$ such that (x, y) is an edge in G_c. In other words, two cells are connected if they contain adjacent points in the continuous graph.

The partition of the space into cells should be maintained in a distributed manner. When a Join operation is performed an existing cell splits, when a Leave operation is performed two cells are merged into one. In our view the task of designing a dynamic and scalable network should follow the following design rules:

1. Choose a proper continuous graph G_c over the continuous space I. Design (and prove the correctness of) the algorithms in the *continuous* setting. Designing the algorithms in the continuous graph is typically quite simple. It has the advantage of being oblivious to the scalability issue, thus bypasses many of the technical difficulties causes by taking the scalability into account. It also offers strong and simple mathematical tools for proving statements.

2. Find an efficient way to *discretisize* the continuous graph in a distributed manner, such that the algorithms designed for the continuous graph would perform well in the discrete graph. The discretization is done via a decomposition of I into cells. An important parameter of the decomposition of I is the ratio between the size of the largest and the smallest cell which we call the *smoothness*. We show that a decomposition in which the smoothness is constant can be used to build the Distance Halving DHT and the expander based networks. We show that if the cells which compose I are allowed to *overlap* then the resulting graph would be fault tolerant.

The main advantage of the approach is that it gives a *unified* method for performing the Join/Leave operation and dealing with the scalability issue, thus separating it from the actual network and allowing a more *modular* design. Of course the proof of the pudding is in the eating. In the following chapters we demonstrate the usefulness of the approach.

1.2 Thesis Organization

The first part of the Thesis consists of Chapters 2 and 3. In it we present and investigate the Continuous-Discrete approach for designing dynamic data structures. The Continuous-Discrete approach is applied for designing a simple and efficient Distributed Hash Table called *Distance Halving*. Chapter 2 presents this construction along with many extensions. We show a fault tolerant variant of the Distance Halving DHT, and provide a dynamic caching algorithm which is guaranteed to eliminate hot spots. We show how a generalization of the approach could be used to build a DHT with an expanding topology. Most of the results in Chapter 2 originally appeared in [99] and [101].

In Chapter 3 we show how the Continuous-Discrete approach could be applied to design scalable and dynamic quorum systems. We also provide some insights regarding the complexity of finding live quorums in face of random failures. These results appear in [103] and its earlier version [100].

In the second part of the Thesis we investigate existing data structures. In Chapter 4 we present and analyze the Neighbor-of-Neighbor (NoN) algorithm in Small World Networks and in Skip Graphs. We provide tight upper and lower bounds on the effectiveness of the NoN routing algorithm and of the Greedy routing algorithm. These results were originally published in [88],[102]. Finally in Chapter 5 we show that with high probability Skip Graphs contain a $4-$regular expander. We demonstrate various applications of this fact, including sampling a random node and finding a highly replicated data item. The Chapter is based upon [17].

Chapter 2

Distributed Hash Tables

Summery: In this Chapter we demonstrate the power of the continuous-discrete approach by suggesting two new P2P architectures and various algorithms for them. The first serves as a DHT (Distributed Hash Table) and the other is a dynamic expander networks. The DHT network, which we call Distance Halving allows logarithmic routing and load, while preserving constant degrees. It offers an optimal tradeoff between the degree and the path length in the sense that degree d guarantees a path length of $O(log_d n)$. A major new contribution of this construction is a dynamic caching technique that maintains low load and storage even under the occurrence of hot spots. Our second construction builds a network that is guaranteed to be an expander. The resulting topologies are simple to maintain and implement. Their simplicity makes it easy to modify and add protocols. A small variation yields a DHT which is robust against random (possibly Byzantine) faults. Finally we show that, using our approach, it is possible to construct *any* family of constant degree graphs in a dynamic environment, though with worst parameters.

2.1 Introduction

A *distributed hash table* (DHT) is a giant hash table that is maintained by a large number of servers in a P2P manner. The hash table interface is useful for the implementation of a large variety of tasks, therefore it received a considerable amount of attention from the research community. Previous DHT designs include [109], Tapestry [128], Chord [120], Pastry [115], CAN [113], Kademlia [91], Viceroy [84] and many more. These systems follow the general paradigm of *consistent hashing* [63]: Let I denote the space into which the data item's keys are hashed. The idea is to assign to each node an ID from I as well. Typically $I = \{0,1\}^k$ for some $k > 0$, with out loss of generality we may assume that $I = [0,1)$. Thus the ID's of the n nodes *partition* I into contiguous segments. Assign each node a segment (say the segment that lies between its ID and the next largest ID) and let each node be responsible for storing all the data items with hash values that fall within its assigned segment. The connections in the network are also determined by the ID's of the different nodes. Connections are set such that the system supports a *lookup* protocol that allows nodes to find the node which is responsible for a required hash value, and thus retrieve a data item. The differences between the various DHT's lie in the different ways in which the connections are established, and the different algorithms in which the routing paths are found[1].

[1]There may be various ways in which a lookup service is implemented even when the network is given and fixed. For instance in 'real life' systems, an iterative lookup algorithm may behave very differently from a recursive one. We are interested in the algorithmic/combinatorial nature of the algorithms and ignore such issues.

Lookup Scheme	*path length*	*congestion*	*linkage*
Chord [120]	$\log n$	$(\log n)/n$	$\log n$
Tapestry [128]	$\log n$	$(\log n)/n$	$\log n$
CAN [113]	$dn^{1/d}$	$dn^{1/d-1}$	d
Small Worlds [70]	$\log^2 n$	$(\log^2 n)/n$	$O(1)$
Viceroy [84]	$\log n$	$(\log n)/n$	$O(1)$
Distance Halving (ours)	$\log_d n$ $(2 \leq d \leq \sqrt{n})$	$(\log_d n)/n$	$O(d)$

Table 2.1: Comparison of *expected* performance measures of lookup schemes.

The methodology used in designing these networks could be roughly described as follows: first find a *static family* of graphs in which there are good protocols for performing the desired tasks. Typically designers aim at one of the classical inter-connection networks such as hypercubes and grids. The next task is to show how to construct in a distributed manner a network with a topology that 'approximates' the topology of the static family of graphs. In this sense CAN approximates the $d-$dimensional torus. Chord and Pastry approximate the hypercube and Viceroy approximates the butterfly. The continuous-discrete approach gives a unified technique for performing this. We use the continuous-discrete approach to design DHT based on the De-Bruijn graph which we call *Distance Halving*.

Like any P2P networks, a DHT must specify the properties aforementioned in the Introduction - low cost of Join/Leave, Scalability and so on. Being a hash table, a good DHT should also have the following properties:

Short Lookup path length: The routing path of a lookup should involve as few machines as possible. We aim to minimize the maximum path length in the network.

Low Congestion: No server should be a bottleneck on the performance of the service. The load incurred by lookups routing through the system should be evenly distributed among participating servers.

Fault tolerance: The service should function well after some of its servers/connections fail. We should consider the scenario in which a random subset of the servers fail, and the worst case scenario in which an adversary chooses which servers fail. In each of these cases there are two models to consider. The first is the fail and stop model in which failed servers/connections do not respond at all. The second is a Byzantine model in which failed servers may act in an inconsistent and malicious way.

Dynamic caching: Highly popular data items may cause a bottleneck at and around their location. Relieving the congestion around the hot spot requires the service to support some dynamic caching mechanism, in which the data item is replicated to other servers. We want to allow the maximally congested server in the system to have a low load while maintaining the number of data items each server has to store as small as possible.

Table 2.1 summarizes the performance of different constructions under these parameters.

2.1.1 Our Contributions

Our Contributions are both in the conceptual and in the concrete levels. Conceptually we provide a set of design rules, and a framework in which we believe it is relatively simple to design and analyze dynamic data structures. We call the approach 'continuous-discrete', it is the first attempt to unify different constructions in this field.

Concretely we suggest six novel constructions and algorithms:

The Distance Halving DHT: We present a novel construction for a DHT in Section 2.2. The construction is very simple and offers logarithmic dilation and load. An important feature is that it has an optimal tradeoff between the degree and the dilation. A degree of d guaranties a dilation of $O(\log_d n)$. Previous constructions either had a logarithmic degree (such as Chord [120]) or were relatively more involved (such as Viceroy [84]). See Table 2.1.

Our DHT construction is inspired by the De-Bruijn graph. We are not the only ones to use the De-Bruijn graph in this context. Constructions using it were suggested independently by Fraigniaud and Gauron [41], Kaashoek and Karger [62] and Abraham *et al* [2]. The parameters they achieve are similar ours, yet their approach is completely different. In particular it is not clear how to obtain in theses constructions the caching protocol and the low load in permutation routing, which we show in the following Sections.

A fault tolerant DHT: In the continuous-discrete approach the ID's of nodes are used to divide the name space into segments, thus converting the continuous graph into a discrete one. In Section 2.3 we show that a different discretization technique which allows segments to *overlap* results in a more fault tolerant construction.

Dealing with hot spots: We show a dynamic caching algorithm in Section 2.4, that provably ensures that under *any* set of requests for data items, all nodes enjoy low load, thus it relieves the occurrence of hot spots. Dynamic caching achieved a considerable amount of attention under many different models. The problem of dynamic caching in DHT's was specifically raised by Ratnasamy *et al* at [114]. To the best of our knowledge the algorithm we present is the first to ensure this property.

Load Balancing: In Section 2.6 we present several algorithms for maintaining a good load balance between nodes. These techniques allow us to build DHT's with constant degrees *with high probability*. One of the methods we show guarantees constant degrees in the worst case. The construction in [41] has maximum out-degree of $\log n$, and so is the maximum in-degree of [62]. The techniques we show are applicable for other DHT constructions as well.

Building expander graphs: In Section 2.7 we show a distributed construction of a network which is guaranteed to be a constant degree expander. Our construction is based upon applying the continuous-discrete approach over the Gabber-Galil [44] continuous graph. Law and Siu have independently designed another algorithm which builds an expander with high probability [73]. Their approach is completely different from ours. Possible applications for dynamic expanders include load balancing jobs and an infrastructure for maintaining probabilistic quorums.

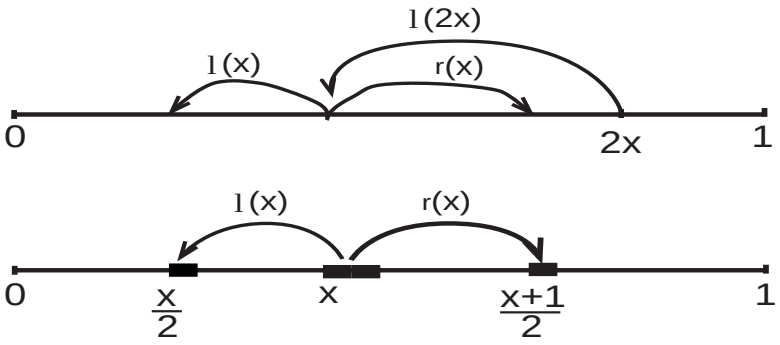


Figure 2.1: The upper figure demonstrates the edges of a point in the continuous graph. The lower figure shows a mapping of a segment to two smaller ones.

Emulating everything: Finally we show in Section 2.5 that the continuous-discrete approach could be used to emulate *any* graph in a distributed setting, including say, a butterfly of a cube connected cycle (albeit with somewhat worst parameters compared to the Distance Halving design).

We stress that the simplicity of the continuous-discrete approach plays a central role in the design and analysis of the algorithms above. In particular in that of the caching algorithm of Section 2.4 and the Fault Tolerant construction of Section 2.3. We see the relative ease in which these problems were solved as a 'proof of concept' for the entire approach.

2.2 The Distance Halving DHT

2.2.1 The Construction

First we define the *continuous* Distance Halving graph G_c. The vertex set of G_c is the interval $I \stackrel{def}{=} [0,1)$. The edge set of G_c is defined by the following functions: $\ell(y) \stackrel{def}{=} \frac{y}{2}$, $r(y) \stackrel{def}{=} \frac{y}{2} + \frac{1}{2}$ where $y \in I$, ℓ abbreviates 'left' and r abbreviates 'right'. Note that the out-degree of each point is 2 while the in-degree is 1. In Figure 2.1 the upper diagram shows the edges of a point in I, the lower diagram shows that an interval is mapped into two intervals, each half its size. We may write $r([y,z)), \ell([y,z))$ meaning the image of the interval $[y,z)$ under r, ℓ. Next we show how to construct a *discrete* Distance Halving graph. The nodes of the graph correspond to a set of n servers $V_0, V_1, \ldots, V_{n-1}$. Denote by $\vec{x}$ a set of n points in I such that $x_0 < x_1 < \ldots < x_{n-1}$. The point x_i would typically be a hash of the i.d of some server V_i. The points of $\vec{x}$ divide I into n segments. Define the *segment* of x_i to be $s(x_i) = [x_i, x_{i+1})$ $(i = 1 \ldots n-1)$ and $s(x_n) = [x_{n-1}, 1) \cup [0, x_1)$.

In the *discrete* Distance Halving graph $G_{\vec{x}}$, each node V_i is associated with the segment $s(x_i)$, we may refer to this segment as $s(V_i)$ as well. If a point y is in $s(x_i)$ we say that V_i *covers* y. A pair of nodes (V_i, V_j) is an edge of $G_{\vec{x}}$ if there exists an edge (y,z) in the *continuous* graph, such that $y \in s(x_i)$ and $z \in s(x_j)$. The edges (V_i, V_{i+1}) and (V_{n-1}, V_0) are added such that $G_{\vec{x}}$ contains a ring. The ring edges are anti-parallel directed edges.

Theorem 2.2.1. *The total number of edges in $G_{\vec{x}}$ without the ring edges is at most $3n-1$.*

Proof. The proof is by induction on n. If $n = 1$ then there are two self edges. Assume $\vec{x}$ has $n-1$ points, and point n is now added. The degree of the continuous graph is 3. The segment that the

new point covers was previously covered by another point, therefore the addition of point n can add at most 3 new edges to the graph. □

Theorem 2.2.1 implies that for every vector $\vec{x}$, the *average* degree of the graph is at most 6. In order to bound the *maximum* degree of the graph, another property should be considered:

Definition 2.2.2. The *smoothness* of $\vec{x}$ is denoted by $\rho(\vec{x})$ and is defined to be $\max_{i,j} \frac{|s(x_i)|}{|s(x_j)|}$.

If it is guaranteed that the smoothness of $\vec{x}$ is bounded by some constant independent of n, we say that $\vec{x}$ is smooth. We may write $\rho(G_{\vec{x}})$ or simply ρ whenever there is no ambiguity. The smoothness of $\vec{x}$ plays a central role in the analysis of the construction.

Theorem 2.2.3. *The maximal out-degree of $G_{\vec{x}}$ without the ring edges is at most $\rho(\vec{x}) + 4$, the maximal in-degree without the ring edges is at most $\lceil 2\rho(\vec{x}) \rceil + 1$.*

Proof. Let i be such that $|s(x_i)|$ is maximal. The length of the minimal segment is therefore at least $\frac{|s(x_i)|}{\rho}$. We have $|r(s(x_i))| = \frac{1}{2}|s(x_i)|$, therefore there are at most

$$\lceil (\tfrac{1}{2}|s(x_i)|)/(\tfrac{|s(x_i)|}{\rho}) \rceil + 1 = \lceil \tfrac{1}{2}\rho \rceil + 1$$

different segments that intersect the interval $r(s(x_i))$. The same argument applies for $\ell(s(x_i))$, and we have $2(\lceil \frac{1}{2}\rho \rceil + 1) \leq \rho + 4$, which bounds the out-degree.

The bound on the in-degree follows in a similar way. Now the preimage of $|s(x)|$ is one contiguous of segment of length $2|s(x)|$, and at most $\lceil 2\rho(\vec{x}) \rceil + 1$ different segments might intersect it. □

Mapping the data items to servers: The mapping of data items to nodes is done in the same manner as other constructions of distributed hash tables (such as consistent hashing [63], Chord [120], Viceroy [84] and CAN [113]). First data items are mapped into the interval I using a hash function. Node V_i should hold all data items mapped to points in $s(V_i)$. We assume that h is some hash function (for instance a $k-$wise independent function for some k), which is chosen at the construction of the system and is given to every server upon joining.

The De-Bruijn Graph The Distance Halving construction resembles the well known De-Bruijn graph.

Definition 2.2.4. The *r-dimensional De-Bruijn graph* consists of 2^r servers and 2^{r+1} directed edges. Each node corresponding to an $r-$bit binary string. There is a directed edge from each node $u_1 u_2 \cdots u_r$ to nodes $u_2 \cdots u_r 0$ and $u_2 \cdots u_r 1$.

The Distance Halving DHT emulates the De-Bruijn graph in the following sense. Assume that $n = 2^r$. Let $\vec{x}$ be a set of m points such that $x_i = \frac{i}{n}$, the discrete Distance Halving graph $G_{\vec{x}}$ without the ring edges is isomorphic to the r-dimensional De-Bruijn graph. To see this blow the interval I by a factor n so that $I = [0, 2^r)$. Now the location of point x_i is at i. Let $v_1 v_2 \ldots v_r$ be the binary representation of i. It is easy to verify that $\ell(x_i)$ is $0v_1 \ldots v_{r-1}$ and that $r(x_i)$ is $1v_1 \ldots v_{r-1}$. Now the isomorphism follows by mapping each $v_1 v_2 \ldots v_r$ of $G_{\vec{x}}$ to $v_r v_{r-1} \ldots v_1$ in the De-Bruijn graph.

The r-dimensional De-Bruijn graph is a well investigated combinatorial object, it is known for instance that its diameter and mixing time are $\Theta(r)$ and that the smallest bisection contains

$\Theta(r^{-1}2^r)$ edges. The ease in which short routes are found makes it a popular topology for parallel algorithm. See Leighton [75] for an overview of various properties of this graph.

While the definition of De-Bruijn graph we presented assumes each node is represented by a binary string, it is natural to generalize the definition so that each node is represented by a string of alphabet size Δ. In this case the diameter of the graph would be $\log_\Delta n$. A variation of the continuous graph emulates the De-Bruijn graph with alphabet size Δ. See more details in Section 2.2.5. Different ways to emulate the De-Bruijn graph in a P2P manner were suggested in [41, 62, 2].

2.2.2 Joining and Leaving the Network

So far we described the graph given a fixed set of points. In the following we show a general algorithmic scheme which allows a new node to join the network. If a new server V wishes to enter the system it does the following:

Algorithm Join

1. Choose some point x and set $V.id \leftarrow x$.
2. Call the lookup procedure[2] for point x. The procedure returns the ID of the node V_j for which $x \in s(x_j)$.
3. The segment $s(x_j)$ should be divided into two parts so that $s(x) = [x, x_{j+1})$. Receive from V_j all the data items that are mapped to $s(x)$ and the addresses of all the neighbors V should have.
4. Inform the neighbors of V so that they can change their address tables accordingly.

How to choose $V.id$ is not shown here. The way nodes choose there location on the ring determines the smoothness of the graph, so the specifics of Step (1) of the scheme are important and may differ according to context. Various options are discussed in-depth in Section 2.6. Alternative ways for performing step (1) were subsequently suggested in many papers, c.f [64],[49],[25],[85],[66].

Step (2) of the algorithm involves one operation of Lookup which would be discussed later. Assuming the graph has constant degree Step (3) of the algorithm involves only $O(1)$ messages.

When node V_i leaves the network, some existing node should take over its segment. The simplest solution would be that the node that is the predecessor of V_i on the ring enlarges its segment such that it includes $s(x_i)$. More sophisticated solutions are discussed in Section 2.6.

2.2.3 The Lookup Operation

We set some notations that would be useful in the future. For any two points $x, y \in I$, define $d(x, y)$ to be $|x - y|$. Let σ denote some infinite sequence of binary digits, and σ_t denote its prefix of length t. Denote by $\sigma_t.0$ and $\sigma_t.1$ the concatenation of a bit to the string σ_t. For every point

[2] We do not deal with the problem of how to perform this initial lookup. It is assumed that a joining node has some host node which helps it to join the system.

$y \in I$ we define the function $w(\sigma_t, y)$ in the following recursive manner:

$$w(\sigma_0, y) = y \tag{2.1}$$
$$w(\sigma_t.0\,,\, y) = \ell(w(\sigma_t, y)) \tag{2.2}$$
$$w(\sigma_t.1\,,\, y) = r(w(\sigma_t, y)) \tag{2.3}$$

In other words $w(\sigma_t, y)$ is the point reached by a walk along the edges of G_c that starts at y and proceeds according to σ_t when 0 represents ℓ and 1 represents r.

Routing properties of the continuous DH graph:

Loyal to the continuous-discrete approach we first demonstrate how routing is performed in the continuous graph. The following observation justifies the name 'Distance Halving':

Observation 2.2.5 (distance halving property). *For all $y, z \in I$ and for all binary strings σ it holds that:*

$$d(r(y), r(z)) = d(\ell(y), \ell(z)) = \tfrac{1}{2} d(y, z) \tag{2.4}$$
$$d(w(\sigma_t, y), w(\sigma_t, z)) \;=\; 2^{-t} \cdot d(y, z) \tag{2.5}$$

Let $\sigma(y)$ be the binary representation of y and $\sigma(y)_t$ the first t bits of $\sigma(y)$. The following claim is used to find short paths between different segments of the continuous graph.

Claim 2.2.6. *Let $y, z \in I$ .For all t it holds that $d\left(y, w(\sigma(y)_t, z)\right) \leq 2^{-t}$.*

The claim states that a walk determined by the binary representation of y would approach y quickly, and this is *independent of the starting point z.*

Proof. Let h_t be the point reached by walking *backwards* (along the in-degree edge) from y for t steps. Such a walk is uniquely defined since each point in I has in-degree of exactly one. Note that the direction (left or right) of the i'th step in this walk is determined by the i'th bit in $\sigma(y)$. In other words it holds that: $w(\sigma(y)_t, h_t) = y$. We have:

$$d\left(w(\sigma(y)_t, z), y\right) = d\left(w(\sigma(y)_t, z), w(\sigma(y)_t, h_t)\right).$$

By Observation 2.2.5 it holds that $d\left(w(\sigma(y)_t, z)\,,\, w(\sigma(y)_t, h_t)\right) = 2^{-t} d(z, h_t) \leq 2^{-t}$. □

The previous two claims demonstrate two ways in which nodes of the continuous graph can approach one another. Loyal to our design rules we use the properties of the continuous graph to design simple routing algorithms on the *discrete* graph. These algorithms emulate the scheme described in Claims 2.2.5 and 2.2.6.

Say node V_i wishes to lookup the point y. The lookup should return the node V_j such that $y \in s(x_j)$. We present two algorithms that perform lookup. The first will have short lookup paths, while the second will increase the lookup path by a factor of at most two and will have better load balancing qualities.

Short Lookup

Claim 2.2.6 states that from any point in I it is possible to reach a point that is close to y by walking according to the binary representation of y. If $y \in s(x_i)$ for some server, then $s(x_i)$ would also contain points close to y. This observation is used to emulate the continuous lookup in $G_{\bar{x}}$. The natural way to use Claim 2.2.6 is to start a walk from some point in $s(x_i)$ according to $\sigma(y)$. The problem with this approach is that the length of the walk should be decided in advance (for every predefined t the walk would lead to a distance of 2^{-t} from y). The alternative we chose is to walk 'backwards'; i.e. walk along a path which starts at y and approaches the point in the middle of $s(x_i)$. This requires the use of the *backward* edges, therefore we assume that every edge is bidirectional (which is the case if channels are established using TCP). The following is a description of the lookup operation initiated by V_i.

Short Lookup (x_i, y)

1. Let z be the point in the middle of $s(x_i)$. Calculate the minimum t such that $w(\sigma(z)_t, y)$ is in $s(x_i)$.
2. Let $h = w(\sigma(z)_t, y)$. Start moving from h (i.e. from $s(x_i)$ and therefore from V_i) on the *backward* edges. After t steps the server covering y is reached.

The length of the lookup path is determined by t. If the segments are smooth, then $|s(x_i)|$ can not be too small thus t must be small as well. A direct corollary of Claim 2.2.6 is that $t \leq -\log(|s(x_i)|)+1$. The shortest segment in I is of length at least $\frac{1}{\rho n}$, therefore we have:

Corollary 2.2.7. *The length of a lookup path taken by* Short Lookup *is at most* $\log n + \log \rho + 1$.

Note that in *Short Lookup* the nodes need not to know what ρ or n is, and all computations are based on local knowledge only. Next we analyze the congestion of *Short Lookup*.

Definition 2.2.8. The *congestion* of node V_i is the probability V_i is active in a routing between a randomly chosen node and a random point in I. The congestion of the network is the maximum congestion over all its nodes.

First we prove that the congestion of the *continuous* graph is low; i.e. the congestion when both source and destination are chosen randomly in I.

Lemma 2.2.9. *Let y, z be chosen randomly from I. The probability that node V participates in a lookup of length t that starts at y and approaches z using the* Short Lookup *algorithm is at most $|s(V)|(t+1)$.*

Proof. Since z is a random point, it holds that $\sigma(z)_t$ is a random sequence of bits. It follows that in order for V to participate in the i^{th} hop of the lookup, two events must occur:

1. There is an interval of length $2^i|s(V)|$ from which it is possible to reach $s(V)$ in i steps. Note that since the in-degree of each point in I is 1, this interval is unequally defined and independent of y, z. It is necessary that y falls *within* that interval.
2. Once y is chosen, it is necessary that the i first bits of the rondom string $\sigma(z)$ would be such that a message starting from y would actually reach $s(V)$.

Both events are independent from one another, therefore for each $i = 0...t$ the probability of server V to participate in lookup in the i^{th} step is $|s(V)| \cdot 2^i \cdot 2^{-i} = |s(V)|$. Adding the probabilities for all $i \leq t$ yields the result. □

Theorem 2.2.10. *The congestion of node V is at most $(\log n + \log \rho + 1)\rho|S(V)|$ which implies that if $\vec{x}$ is a smooth set of points, the congestion of $G_{\vec{x}}$ under* Short Lookup *is $\Theta(\frac{\log n}{n})$.*

Proof. The proof follows the same lines of the previous lemma. The number of hops in the path is bounded by $\log n + \log \rho + 1$. For each hop in the path, we show that the probability that V participates is bounded by $\rho|S(V)|$. Fix some $i \leq \log n + \log \rho + 1$. As before, there is a unique segment of length $2^i|S(V)|$ in which y must exist in order for the path to reach $s(V)$ on i^{th} step. The probability of this occurring is $2^i|s(V)|$. Now, given that y is in this segment, in order for V to participate, there is a unique set of i steps leading from y to $|s(V)|$. Let U be a randomly chosen node which lookups y and let z be its middle point. We need to calculate the probability the first i bits of $\sigma(z)$ are indeed i bits leading from y to $s(V)$. This is tantamount to asking what is the probability z falls within some segment of length 2^{-i}. Since the smoothness is ρ, the size of each $s(U)$ is at least $\frac{1}{\rho n}$, thus there are at most $2^{-i}\rho n$ different nodes whose middle points might be in the segment. Thus the probability V participates in the i^{th} step is bounded by:

$$\frac{2^{-i}\rho n}{n} \cdot 2^i|S(V)| = \rho|S(V)|.$$

The second part of theorem follows by the fact that if ρ is constant then $|s(V)|$ is $\Theta(1/n)$.

□

Note that there is no uncertainty in the result of Theorem 2.2.10. If $\vec{x}$ is smooth then the congestion is low for *all* nodes with certainty.

Distance Halving Lookup

The Distance Halving lookup scheme enjoys small congestion even in a worst case permutation routing. It has two phases, the first phase is to send the message to an almost random destination, the second phase routes the message from the random destination to the target. First we describe how to perform the first phase. When node V_i initiates a lookup for point y, it first chooses a random string of bits τ. The header of the message V_i sends should contain V_i location - x_i, the target y, the random string τ, and a counter t initially set to 0. Upon receiving a message a node does the following:

Distance Halving Lookup - Phase I

1. Check if $w(\tau_t, y)$ is covered by the current segment or by one of the neighboring segments. If so move the message to the server which covers $w(\tau_t, y)$ and move to phase II.

2. Set $t \leftarrow (t + 1)$ and update the header of the message.

3. Send the message to the neighbor covering the point $w(\sigma_t, x_i)$. (An edge must exist).

In the second phase the message moves *backwards* from $w(\tau_t, y)$ to y along the backward edges. The counter t may not be updated, it could be calculated from the source and current location of the message.

It is convenient to think of the routing as if two messages are moving simultaneously, one from the source and one from the target. Both of them move according to the same sequence τ. Claim 2.2.5 states that each step the distance between them is reduces by half, until their distance is at most $\frac{1}{\rho n}$, in which case they would be in the segment of the same node (or neighboring nodes). The following theorem is a direct result of Claim 2.2.5:

Theorem 2.2.11. *The length of a lookup path taken by Distance Halving Lookup is at most* $2\log n + 2\log \rho$.

The following theorem states that the maximum congestion of the Distance Halving Lookup is also logarithmic.

Theorem 2.2.12. *Let $\vec{x}$ be a smooth set of points. The congestion of $G_{\vec{x}}$ under Distance Halving Lookup is* $\Theta(\frac{\log n}{n})$.

Proof. The first phase of the routing is basically similar to the greedy scheme, i.e. the messages passes through a random path. Thus the same analysis of Theorem 2.2.10 follows. The second phase of the routing is analogical to the first phase. □

2.2.4 Permutation Routing

The Distance Halving Lookup is similar in spirit to the routing scheme suggested by Valiant [122] for the hypercube, therefore it is not surprising that it imposes small congestion for worst case permutation routing. Let τ be some permutation and over $[n]$ and assume that for all i node V_i initiates a Distance Halving Lookup for a point in $s(V_{\tau(i)})$.

Theorem 2.2.13. *Given that $G_{\vec{x}}$ is smooth, then for every permutation τ it holds that when routing τ w.h.p each node participates in the routing of at most $O(\log n)$ messages, where the probability is taken over the random choices of the routing algorithm.*

Proof. First, note that since the length of each of the n lookup paths is typically $\Theta(\log n)$, by an averaging argument there would be a node which served $\Omega(\log n)$ messages, so the Theorem is tight.

Fix a node V. In the following we prove that the *expected* number of lookups that V participates in is $O(\log n)$. The high probability bound would follow later. For a contiguous segment $Q \subset I$, let L_i^Q be the random variable counting the number of lookups that reach segment Q at the i^{th} step[3], and consider only the first phase of the routing. We claim that

$$\mathbb{E}\left[L_i^{s(V)}\right] \leq n(\rho+1)|s(V)|.$$

We prove it by induction on i. As common in proofs by induction we need to strengthen the claim slightly and prove that *any* segment $Q \subset I$, has $\Theta(n(\rho+1)|Q|)$ messages reaching it at step i. Clearly Q can cover points which belong to at most $n\rho|Q| + 1 \leq n(\rho+1)|Q|$ different servers, therefore $L_0^Q \leq n(\rho+1)|Q|$.

[3]It is not necessary that the messages in the i^{th} step indeed reach the segment together. There is no implied assumption of synchrony.

For a message to reach Q in the i^{th} step, it must be that on step $i-1$ the message was in an interval of length $2|Q|$ and then moved to Q. Call this interval Q'.

$$\mathbb{E}\left[L_i^Q\right] = \frac{1}{2}L_{i-1}^{Q'}$$
$$\mathbb{E}\left[L_i^Q\right] = \frac{1}{2}\mathbb{E}\left[L_{i-1}^{Q'}\right]$$

The induction hypothesis states that

$$\frac{1}{2}\mathbb{E}(L_{i-1}^{Q'}) \leq \frac{1}{2}n(\rho+1)|Q'| = n(\rho+1)|Q|.$$

There are at most $\log n + \log \rho$ steps in the first phase, and $|s(V)| \leq \frac{\rho}{n}$. Summing over i yields that the expected number of lookups that pass through V in the first phase is at most $(\log n + \log \rho)(\rho + 1)\rho$, which is $O(\log n)$ when ρ is a constant.

The second phase of the routing is similar to the first phase but in reverse order. When paths are seen as going from the target towards the middle random point, then each path is a random walk determined by the random choices of the routing algorithm. Thus the analysis of the first phase holds for the second phase as well. We conclude that if $\vec{x}$ is smooth, then each node participates in the routing of expected $O(\log n)$ messages.

Next we prove that w.h.p the actual number of lookups that pass through V is indeed $O(\log n)$. Let p_i $(i = 1 \ldots n)$ indicate the probability that message i passes through $s(V)$ during the first phase of the lookup. By the previous claim we know that $\sum_{i=1}^{n} p_i \in O(\log n)$. The random choices that determine the paths that messages take are independent from one another. Standard use of Chernoff's inequality yields:

$$\Pr\left[\sum p_i > \Theta((1+\epsilon)\log n)\right] \leq n^{-\Theta(\epsilon^2)}$$

Choosing ϵ large enough would allow us to use the union bound over all servers thus proving the bound for the first phase. As before the analysis of the second phase is similar to that of the first. □

In reality, permutation routing is not likely to occur. Rather, nodes initiate lookups for data items, and the hash function which maps data items to points should spread them evenly along I. A family of hash functions $\mathcal{H}$ is said to be *k-wise independent* if when $h \in \mathcal{H}$ is chosen randomly it holds that for any data items $m_1 \neq m_2 \neq \ldots \neq m_k$ the random variables $h(m_1), h(m_2), \ldots, h(m_k)$ are independent and uniformly distributed in I.

Next we handle the scenario where each node V_i initiates a lookup for data item m_i; i.e. node V_i seeks for the node covering the point $h(m_i)$. We also assume that $m_i \neq m_j$ for all $i \neq j$.[4]

Theorem 2.2.14. *Given that $G_{\vec{x}}$ is smooth and h is $\log n$-wise independent, for every permutation τ it holds that when routing τ with high probability each node participates in the routing of $O(\log n)$ messages, where the probability is taken over the choice of the hash function and the random choices of the routing algorithm.*

[4]Section 2.4 deals with the case where the same item is queried by multiple nodes.

Proof. The proof follows the same lines as the proof of Theorem 2.2.13. The only difference is that in the second phase it is not true that the paths messages take are independent, but rather are $\log n$-wise independent. Fix some node V and let X_i denote the event that message i passed through V in the second phase of the routing. The analysis of Theorem 2.2.13 holds in this case as well so we know $E[\sum X_i]$ is $O(\log n)$ and that the X_is are $\log n$-wise independent. We use the following high moment version of Chebyshev's inequality:

Claim 2.2.15. *Let $X_1, X_2, \ldots, X_n$ be $2k-$wise independent random variables. Denote by $X = \sum_i X_i$, $\mu = E[X]$ and $\epsilon > 0$. If $Var[X] \leq E[X]$ then:*

$$\Pr[|X - \mu| \geq \epsilon\mu] \leq (\epsilon^2\mu)^{-k}$$

In our case the random variables are Bernoulli variables so $Var[X_i] \leq E[X_i]$. The pair-wise idependence implies that $Var[X] \leq E[X]$. Now we apply Claim 2.2.15. Setting $2k = \log n$ and ϵ large enough the probability a node handles more than $\epsilon \log n$ messages is smaller that $\frac{1}{n^2}$ and a union bound over all nodes completes the proof of the Theorem. □

Theorem 2.2.14 demonstrates that in some sense the Distance Halving Lookup is good in a worst case scenario. An adversary may choose for each V_i its appropriate m_i (as long as the adversary is oblivious of h). It is worth noting a few facts:

- Strictly speaking, the Distance Halving routing is *not* an oblivious routing; i.e. the edge in which the message is sent, is not a function of the destination only but is also a function of the numeric value of x_i and the random string τ.
- The routing algorithm is sensitive to small perturbations in the numerical value of the parameters. It is important to be precise enough, and to allocate enough bits for the variables.

2.2.5 Degree-Path Length Optimality

Increasing the degree may reduce the lookup length and the congestion. For any $\Delta \geq 2$ construct a continuous graph with the following edges: $f_i(y) = \frac{y}{\Delta} + \frac{i}{\Delta}$ $(i = 0, 1, \ldots, \Delta - 1)$. Now Claim 2.2.5 translates to be $d(f_i(y), f_i(z)) = \frac{1}{\Delta} d(y, z)$ and Claim 2.2.6 translates to be $d(y, \sigma_t(z)) \leq \Delta^{-t}$. Therefore:

Theorem 2.2.16. *A discretization of the continuous graph would result with a graph of degree $\Theta(\Delta)$ and with path length $\Theta(\log_\Delta n)$.*

Note that the diameter of any graph with degree Δ is at least $\log_\Delta n$, so for every Δ the construction has optimal path length with respect to the degree (up to constants). Two interesting options are setting $\Delta = \log n$ or $\Delta = n^\epsilon$ (for some constant ϵ), as the first results with a lookup length of $\frac{\log n}{\log \log n}$, and the second with a lookup length of $O(1)$. It is worth noting that the analysis of previous Sections generalizes so that for each choice of Δ, if the points are smooth the congestion would be $\Theta(\frac{\log_\Delta n}{n})$, thus the increase of the degree would decrease the congestion.

2.3 Fault Tolerant constructions

In this section we present a fault tolerant DHT. There are two common methods for modeling the occurrence of faults. The first is the random fault model, in which every node becomes faulty with some probability and independently from the other nodes. The other is the worst case model in which an adversary which knows the state of the system chooses the faulty subset of nodes. There are several models that describe the behavior of faulty nodes. One of them is the fail-stop model in which a faulty node is effectively being deleted from the system. Another is a spam generating model in which a faulty node may produce arbitrary false versions of the data item requested. A third model is the Byzantine model in which there are no restrictions over the behavior of faulty node.

Our construction is based on the DH *continuous* graph, described in the previous Section. It differs from the construction of Section 2.2, only in the discretization, by letting the segments of the vertices *overlap*. In the random fault model, if we want all servers to be able to access all the data items then it is necessary that the degree be at least $\Omega(\log n)$ and that every data item is stored by at least $\Omega(\log n)$ servers. Otherwise with high probability there would be servers disconnected from the system and data items completely deleted from the system. Indeed our construction has logarithmic degree. We show two routing algorithms. The first has time and message complexity of $O(\log n)$. It guarantees that in the random fail-stop model w.h.p *all* nodes can locate *all* data items. The second routing algorithm guarantees the same but under the random spam generating model. This algorithm has running time (parallel) of $O(\log n)$ and message complexity of $O(\log^3 n)$.

2.3.1 Related Work

Several peer-to-peer systems are known to be robust under random deletions ([128], [120], [113]). Stoica *et al* prove that the Chord system [120] is resilient to random faults in the fail-stop model, Hildrum and Kubiatowicz [60] proved the resilience of Pastry and Tapestry. It does not seem likely that these systems could be made spam resistant without a significant change in their design. Fiat *et al* [116, 40] propose a content addressable network that is robust against deletion and spam in the *worst case* scenario, i.e. when an adversary can choose which nodes fail. In this model some constant fraction of the non-failed nodes could be denied from accessing some of the data items. While their solution handles a more difficult model than ours, it has several disadvantages:

- It is not clear whether the system can preserve its qualities when nodes join and leave dynamically.
- The linkage needed is $\Omega(\log^2 n)$.
- The construction is very complicated.

It is important to note that all construction (including ours) assume that the construction itself was done properly; i.e. that during the Join/Leave operations nodes followed the protocol. Giving any sort of performance guarantees when nodes are allowed arbitrary behavior during network construction seems to be a very difficult task.

2.3.2 The Overlapping Distance Halving DHT

Our construction (yet again) is a discretization of a continuous graph. The continuous graph we use is the *same* continuous graph used to build the DH DHT in Section 2.2.1. The difference is in

the discretization technique.

Each node V_i $(1 \leq i \leq n)$ in the graph is associated with a *segment* $s(V_i) \stackrel{def}{=} [x_i, y_i]$. These segments should have the following properties:

Property I - The set of points $\vec{x} = x_1, x_2, \ldots, x_n$ is evenly distributed along I. Specifically we desire that every interval of length $\frac{\log n}{n}$ contains $\Theta(\log n)$ points from $\vec{x}$. The point x_i is fixed and would not change as long as V_i is in the network.

Property II - The point y_i is chosen such that $|x_i - y_i| \in \Theta(\frac{\log n}{n})$. It is important to notice that for $i \neq j$, $s(V_i)$ and $s(V_j)$ may *overlap*. The point y_i would be updated as nodes join and leave the system and the value of n changes. The precise manner in which y_i is chosen and updated would be described in the next section.

The edge set of G is defined as follows. A pair of of nodes (V_i, V_j) is an edge in G if $s(V_i)$ and $s(V_j)$ are connected in G_c or if $s(V_i)$ and $s(V_j)$ overlap. The edges of G are bi-directional. As before, a point $z \in I$ is said to be *covered* by V_i if $z \in s(V_i)$. The mapping of data items to nodes is done as before, node V_i stores all data items $\mathcal{D}$ for which $h(\mathcal{D}) \in s(V_i)$ when h is some hash function. We observe the following:

1. Each point in I is covered by $\Theta(\log n)$ nodes of G. This means that each data item is replicated $\Theta(\log n)$ times.

2. Each node in G has degree $\Theta(\log n)$.

Join and Leave: Our goal in designing the Join and Leave operations is to make sure that properties I,II remain valid w.h.p. When node V_i wishes to join the system it does the following:

Join Algorithm V_i

1. Choose at random $x_i \in [0, 1)$

2. Calculate a variable q_i which is an estimation of $\frac{\log n}{n}$.

3. Set $y_i = x_i + q_i \mod 1$.

4. Update all the appropriate neighbors according to the definition of the construction.

5. The neighbors may decide to update their estimation of $\frac{\log n}{n}$ and therefore change their y value.

When node V_i wishes to leave the system (or is detected as down) all its neighbors should update their routing tables and check whether their estimation of $\frac{\log n}{n}$ should change. If so they should change their y value accordingly. The following lemma is straight forward:

Lemma 2.3.1. *If n points are chosen randomly, uniformly and independently from the interval $[0, 1]$ then with probability $1 - \frac{1}{n}$ each interval of length $\Theta(\frac{\log n}{n})$ would contain $\Theta(\log n)$ points.*

As a consequence if each node chooses its x-value uniformly at random from I then property-I holds. Observe that if each node's estimation of $\frac{\log n}{n}$ is accurate within a multiplicative factor then property II holds as well. The procedure for calculating q_i is very simple. Assume x_j is the predecessor of x_i along I. It is proven in [84] that with high probability

$$\log n - \log\log n - 1 \leq \log\left(\frac{1}{d(x_i, x_j)}\right) \leq 3\log n$$

Conclude that node V_i can estimate $\log n$ within a multiplicative factor simply by inverting the distance between its x-value and the x-value of its predecessor. Call this estimation α_i. A multiplicative estimation of $\log n$ implies a *polynomial* estimation of n, therefore an additional idea should be used. Let q_i be such that in the interval $[x_i, x_i + q_i]$ there are *exactly* α_i different x-values. A direct consequence of Lemma 2.3.1 is the following:

Lemma 2.3.2. *With high probability the number q_i estimates $\frac{\log n}{n}$ within a multiplicative factor.*

When a node joins or leaves the system at most $O(\log n)$ nodes need to updates their q value. So with high probability property II holds as well.

Mapping the data items to servers: The mapping of data items to servers is done in the same manner as previously. First data items are mapped into the interval I using a hash function. Node V_i should hold all data items mapped to points in $s(V_i)$. Note that all nodes holding the same data item are connected to one another so they form a clique. If a node storing a data item is located, then other nodes storing the same data item are quickly located as well. This means that accessing different nodes associated with the same data item in parallel can be simple and efficient. It suggests storing the data using an erasure correcting code, (for instance the digital fountains suggested by Byers *et al* [26]) and thus avoid the need for replication. The data stored by any small subset of the servers would suffice to reconstruct the data item. Weatherspoon and Kubiatowicz [124] claim that an erasure correcting code may improve significantly the bandwidth and storage used by the system.

2.3.3 The Lookup Operation

The routing properties of the continuous graph G_c were discussed in Section 2.2.3. Assume processor V_i lookups point $y \in I$. Recall that Claim 2.2.6 implies that there is a point $z \in s(V_i)$ such that there is a path of length $O(\log n)$ between z and y in G_c. Call this path the *canonical path*. The canonical path exists is G_c, yet by the definition of G, if (a, b) is an edge in G_c, a is covered by V_i and b is covered by V_j then the edge (V_i, V_j) exists in G. This means that the canonical path can be *emulated* by G.

Simple Lookup: Every point in I is covered by $\Theta(\log n)$ nodes. When node i wishes to pass a message to a node covering point $z \in I$ it has $\Theta(\log n)$ *different* neighbors that cover z. In the *Simple Lookup* it chooses *one* of these nodes at random and sends the message to it. The path of the message could be determined either by *Short Lookup* or by *Disntance Halving Lookup*. Either way, the following theorem follows directly from Claim 2.2.6 and Theorem 2.2.10.

Theorem 2.3.3. *Simple Lookup has the following properties:*

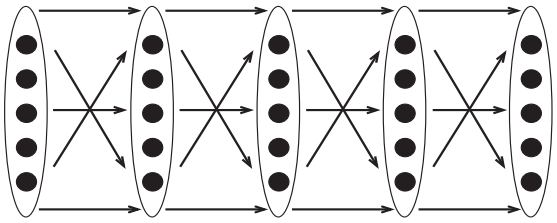


Figure 2.2: In the spam resistance protocol the message is sent through *all* the nodes covering the path.

1. *The length of each lookup path is at most* $\log n + O(1)$*. The message complexity is* $\log n + O(1)$*.*
2. *If* V_i *is a randomly selected node and* y *is chosen at random from* I*, then the probability a given node participates in the lookup is* $\Theta(\frac{\log n}{n})$*.*

The following theorem describes the fault tolerance properties of the lookup:

Theorem 2.3.4. *If each node fails independently with fixed probability* p*, then for sufficiently small* p *(which depends entirely on the parameters chosen when constructing* G*), with high probability each surviving node can locate all data items.*

Proof. We prove the following claim:

Claim 2.3.5. *If* p *is small enough, then w.h.p every point in* I *is covered by at least one server.*

Proof. Assume for convenience that $x_1 < x_2 < \cdots < x_n$. Each point in an interval $[x_i, x_{i+1}]$ is covered by the *same* set of $\Theta(\log n)$ nodes. Call this set S_i. We have

$$\Pr[\text{ All nodes in } S_i \text{ were deleted }] = p^{\Theta(\log n)}$$

Therefore for sufficiently small p this probability is smaller than n^{-2}. Applying the union bound over all i yields that with probability greater than $1 - \frac{1}{n}$ every point in I is covered by at least one node. It is important to notice that for an arbitrary value of p it is possible to adjust the q values, so that each point in I is covered by sufficiently many servers, and the claim follows. □

For every edge (a, b) in G_c there exists at least one edge (V_i, V_j) in G such that V_i covers a and V_j covers b, therefore the path could be emulated in G and the simple lookup succeeds. We stress that after the deletions the lookup still takes $\log n$ time and $\log n$ messages. Furthermore the average load induced on each node increases only by a constant factor. □

Spam Resistant Lookup: Now we assume that a failed node may generate arbitrarily false data items. We show that every node can find all *correct* data items w.h.p. Just as in the simple lookup, the spam resistant lookup between node V and $y \in I$ emulates a path between $s(V)$ and y in G_c. The main difference is that now when node V_i wishes to pass a message to a node covering point a it will pass the message to *all* $\Theta(\log n)$ servers covering a. At each time step each node receives $\Theta(\log n)$ messages, one from each node covering the previous point of the path. The node sends on a message only if it were sent to it by a *majority* of nodes in the previous step.

Theorem 2.3.6. *Assume each node fails with some small enough probability p. The spam resistant lookup has the following properties:*

1. *With high probability all surviving nodes can obtain all correct data items.*
2. *The lookup takes (parallel) time of* $\log n$.
3. *The lookup requires* $O(\log^3 n)$ *messages in total.*

Proof. As before, Statement (2) follows directly from Claim 2.2.6. Statement (3) is correct since each edge of the path in G_c translates to a bipartite complete graph with $O(\log n)$ nodes and $O(\log^2)$ edges, and a message is passed along each of these $O(\log^2)$ edges. It remains to prove Statement (1). Claim 2.3.5 in fact shows that If each node fails with probability p, then for sufficiently small p (which depends entirely on the parameters chosen when constructing G) it holds that with high probability every point in I is covered by a majority of non-failed nodes. Now the proof of Theorem 2.3.6 is straight forward and is done by induction on the length of the path. Every point along the path is covered by a majority of good nodes, therefore every node along the path would receive a majority of the authentic message. It follows that with high probability *all* nodes can find *all* true data items. □

The easy proofs of Theorems 2.3.4 and 2.3.6 demonstrate the advantage of designing the algorithms in G_c and then migrating them to G. Proving the robustness of G_c is a straight forward argument.

2.4 Dynamic Caching - Relieving Hot Spots

In this section we discuss a protocol that eliminates the occurrence of hot spots in the network. A *hot spot* occurs whenever a data item is requested simultaneously by a large number of clients - an event that happens quite often in today's internet. A highly popular data item might not only cause the server holding it to be swamped, but might also cause a bottleneck at and around its location. In order to avoid the congestion caused by a hotspot it is necessary to replicate the popular data item to other servers (i.e. caching), such that the load of handling all requests is distributed between a large number of servers. Relieving hot spots was pointed out as one of the main open problems regarding the design of DHT's by Ratnasamy *et al* [114]. To the best of our knowledge ours is the first protocol that resolves this problem, at least in the sense of providing a provable guarantee. A detailed comparison with previous work is provided bellow. A dynamic caching protocol should satisfy four properties:

1. *Prevent Swamping:* Each server should handle as few messages as possible. This is the ultimate goal of the protocol, and should hold for every possible set of requests.
2. *Keep the Caches small:* Each server has a cache in which it stores data items. A trivial, yet inefficient, caching protocol would be to store all data items in all servers. Such a solution would of course prevent swamping, but would have horrific performance in terms of memory use. Our goal is to keep the cache of each server as small as possible.
3. *Reduce latency:* The caching protocol may cause some delays in obtaining the desired data item. Our goal is to reduce this delay to a minimum.

4. *Keep update time low:* Content may change over time, a caching protocol must be able to accommodate efficiently changes in the cached data item itself.

Previous Work

Various caching techniques were suggested in the literature, which operate under various distributed models (e.g. [109],[63],[32],[29]). Chankhunthod *et al* [29] suggested that caches be arranged as trees, where a request for the data item arrives at a random leaf of the tree and if possible handled there. If a cache does not hold the data item (and neither does its sibling) it passes the request to its parent in the cache tree until finally the message is forwarded to the root which must hold the item. The advantage of the cache tree is that requests are divided more or less evenly between the leaves and each cache receives requests from its children only, thus no cache is being swamped. Karger *et al* [63] enhanced the idea and suggested that each data item have a different random tree of caches, thus better dividing the load. They used hash functions in order to sample a cache tree for each data item, and managed to prove that a single popular data item will not cause a hot spot with high probability.

In the context of a P2P overlay network where each node in the network plays both the role of the server (with its cache) and the role of the client, the suggestion of Karger *et al* [63] has the following disadvantage. The random cache trees associated with each data item are not supported by the overlay network. Thus, when a cache needs to forward a request to its parent it must either use a DHT lookup or maintain a separate overlay network for this use only. The first alternative would cause a meaningful slowdown and has a high communication complexity. The second alternative would increase the linkage of each node and the maintenance cost associated with it. We also note that the additional messages that run in the system due to the caching protocol may cause congestion in their own sake.

Our Contribution

Our suggestion is similar in spirit to that of [63], we also construct a random tree of caches for each data item and forward a message from a random leaf towards the root. Our scheme differs in one important point: we use the Distance Halving overlay network as the overlay network of the caching protocol too; i.e. we couple the cache trees with the Distance Halving graph. The Distance Halving routing algorithm ensures that requests arrive at a random leaf of the cache tree. Thus the caching protocol requires no extra connections and imposes no extra delay. As a result we are able to show that any set of requests (for possibly many data items) do not swamp servers and do not cause congestion.

Given a 'batch' of n requests for data items that arrive in the network[5], the main achievement of this section is to show a dynamic caching protocol with the following properties:

- *Swamp Prevention:* Each server's cache is hit $O(1)$ times on average and $O(\log^2 n)$ with high probability. The total number of messages passed through each server (by the routing scheme and by the caching scheme) is $O(\log^2 n)$ w.h.p..
- *Small Caches:* With high probability each server stores at most $O(\log n)$ data items in its cache. There are at most $O(\frac{n}{\log n})$ new copies of data items stored in caches throughout the network.

[5]Since n denotes the number of servers in the network, we assume that each server issues one request.

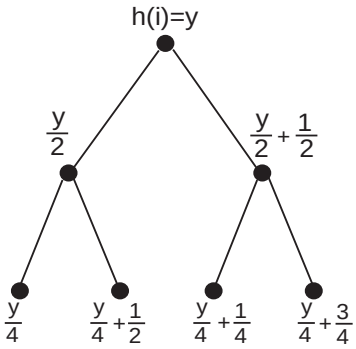


Figure 2.3: The first 2 layers of the path tree.

- *No Caching Latency:* The caching protocol causes *no* extra delay.
- *Quick Content Update:* Changes in the data item are updated in $O(\log n)$ time.

2.4.1 The Protocol

As usual we first describe the protocol in the continuous graph. Denote by $\mathcal{D}$ the set of data items. Let $i \in \mathcal{D}$ be a popular data item and denote y to be $h(i)$, so the server which covers y also holds a copy of i.

Definition 2.4.1. The *path tree* rooted at y is a subgraph of the the continuous graph G_c. The tree is created by assigning y as the root, and each node z in the tree is the parent of $\ell(z), r(z)$.

Here we use the term *nodes* only in the context of the the path tree, and the term *servers* to describe the actual entities in the network. The first two layers of the path tree are drawn in Figure 2.3. The key observation is that if the Distance Halving routing algorithm (of Section 2.2.3) is used , then every request for i would reach y via a *random path* in the path tree; i.e. the probability that a message reaches y via the point $\frac{y}{2}$ is half, and so on. This observation suggests that it is wise to replicate data item i from the root of the tree downward, thus creating a cache tree (a-la [29],[63]). If the data item is copied into the nodes of some layer, then the randomness of the routing protocol ensures that requests would be divided evenly (more or less) among the nodes. In other words, the continuous graph may serve as a random *cache tree* where the nodes of the path tree hold the data item. We call a node that holds a copy of the data item i an *active node*. The tree which consists of all the active nodes is the *active tree*.

In order to deal with a dynamic caching setting formally, we need to define the dynamics of the requests. Assume all servers of the system count[6] the same time epoch. Each server decides upon some number c that is a parameter of the protocol and would serve as a threshold. Typically c would be in the order of $\log n$ and may be updated over time. It is not necessary that all servers choose the exact same parameter, yet for sake of convenience we assume that c is a global parameter known to all servers.

[6] We don't assume that the system is synchronized, this assumption is for convenience and does not play a major role.

Continuous Hot Spots Protocol:

1. Each leaf of the active tree holds a counter which indicates the number of times the data item was requested during an epoch. Once a data item is requested more than c times, the leaf replicates the data item into both its children, effectively blocking itself from handling subsequent requests.

2. If z is the parent of two leaves of the active tree, then at the end of an epoch z performs the following procedure: It checks how many times i was supplied by its children during the epoch. If i was supplied less than c times by both its children, then both children may delete the item i and cease to be active. As a consequence the point z becomes a leaf of the active tree.

3. Step 2 repeats itself recursively, in the same epoch, collapsing the active tree if there are no requests.

In practice it may be beneficial to set a *different* threshold in Step (1) and Step (2). This would add stability to the active tree when the rate of requests is close to the threshold. It also may be more efficient to modify Step (1) such that the data item is initially copied into *one* child, and after another c requests into the second. While both modifications may increase efficiency slightly, they also complicate the analysis.

Presentation: In Section 2.4.2 we analyze the protocol on the continuous graph, i.e. on the active tree. In Section 2.4.3 we analyze the case of a single hot spot. In Section 2.4.4 we analyze the more general case, in which multiple hot spots are formed.

2.4.2 The Active Tree

Denote by q_i the number of times i is requested during an epoch. Each node of the active tree handles at most c requests. Two siblings are deleted if they handled less than c requests each, therefore:

Observation 2.4.2. *For every initial active tree, by the end of the epoch the active tree contains at most $\frac{4q_i}{c}$ nodes. Therefore, the total size of caches in the network is at most $\frac{4q_i}{c}$.*

Observation 2.4.3. *The distance between two points in the j^{th} layer of a path tree is at least 2^{-j}.*

Next we show how the active tree grows and shrinks as a function of q_i. The threshold c is assumed to be at least $\log n$.

Lemma 2.4.4. *If $c \geq \log n$ then with probability at least $1 - \frac{1}{n}$, the lowest point in the active tree at the end of the epoch is at layer at most $\log(q/c) + O(1)$, where the probability is taken over the random decisions of the Distance Halving routing scheme.*

Proof. If for some layer j of the tree it holds that each node in the layer receives less than c messages, then clearly the tree cannot exceed layer j. Consider layer $j = \log(q/c) + t$ of the tree, for some constant t. There are $\frac{q}{c}2^t$ nodes in layer j. The randomness of the routing algorithm causes each message to reach the target independently through a random node in the layer. A standard balls and bins argument shows that with high probability each node in level j received at most c

messages: The probability a message reaches a fixed node in level j is 2^{-j}. The total number of messages reaching the node is distributed according to the Binomial distribution with expectation $2^{-j}q = 2^{-t}c$. Chernoff's bound (e.g. [13]) states that for a binomial variable X it holds that $\Pr[|X - \mu(X)| \geq \epsilon\mu(X)] \leq 2e^{-\delta\epsilon\log\epsilon\mu(X)}$ where $\epsilon > 10$ and δ is a constant independent of ϵ, n. Substituting $\mu(X)$ for $2^{-t}c$ and ϵ for 2^t we get that the probability the number of messages handled by a node exceeds c is at most $e^{-\Theta(tc)}$. If t is large enough and c is $\Omega(\log n)$, then with probability at most $\frac{1}{n^2}$ the node receives at most c messages. Union bounding over the 2^j nodes in the layer yields that with probability at least $1 - \frac{1}{n}$ all nodes of layer j receives less than c messages. □

Lemma 2.4.5. *The load on each active node is bounded as follows:*

1. *The cache at each node is hit at most c times.*
2. *Given that c is $\Omega(\log n)$, w.h.p each active node passes at most $O(\log n)$ messages up to its parent where the probability is taken over the random choices of the routing algorithm.*

Proof. Once a cache is hit more than c times in the same epoch the node replicates the data item to its children, effectively blocking it from being hit again. To prove the second claim consider a node at level $j+1$ and denote by X the number of messages it passed on to its parent. There are at most 2^j active nodes in the first j levels of the tree, therefore at most $c2^j$ requests were passed to the first j levels of the tree by the 2^{j+1} nodes in level $j+1$, so X has the binomial distribution with expectation at most $O(c/2)$. Now the analysis is similar to that of Lemma 2.4.4. Chernoff's bound states that $\Pr[|X - \frac{c}{2}| \geq \epsilon\frac{c}{2}] \leq e^{-\Theta(\epsilon\log\epsilon\frac{c}{2})}$. Since c is $\Omega(\log n)$ we have that for ϵ large enough, with probability $1 - \frac{1}{n}$ the number of messages passed on by each of the nodes is $O(\log n)$. □

2.4.3 Analysis of a Single Hotspot

In the discrete protocol (as usual) server V emulates all the points in $s(V)$. Lemma 2.4.5 bounds the load of each node in the active tree, so all we need is to calculate how many nodes of the active tree V covers. Figure 2.4 demonstrates a mapping from some active tree to the servers.

Theorem 2.4.6. *Server V covers with high probability $O(\log(q/c) + \frac{q}{c}|s(V)|)$ nodes of the active tree, where the probability is over the random choices of the routing algorithm.*

Proof. Observation 2.4.3 implies that server V covers at most $\lceil|s(V)|2^j\rceil$ points from layer j. It may be that V covers one point from each layer of the tree. If for instance V covers the point 0 and the root of the tree, then it also covers all the points in the left branch of the tree. Summing over the levels of the tree Lemma 2.4.4 implies that the number of times V served as a cache is w.h.p.

$$\sum_{j=0}^{\log q/c+O(1)} \left(\lceil|s(V)|2^j\rceil\right) \leq \sum_{j=0}^{\log q/c+O(1)} |s(V)|2^j + 1 = O(\log(q/c) + \frac{q}{c}|s(V)|)$$

□

The Theorem states that the number of times V served as a cache is $O(c\log(q/c) + q|s(V)|)$. Given that c is $\Theta(\log n)$ and $q \leq n$ (each server may issue at most one request per time epoch), and $|s(V)| \leq \frac{\log^2 n}{n}$ then the bound translates to $O(\log^2 n)$.

Note that Theorem 2.4.6 does not assume that the hash function h has any specific properties and it holds even if an adversary is allowed to choose $h(i)$. In the following we assume the hash

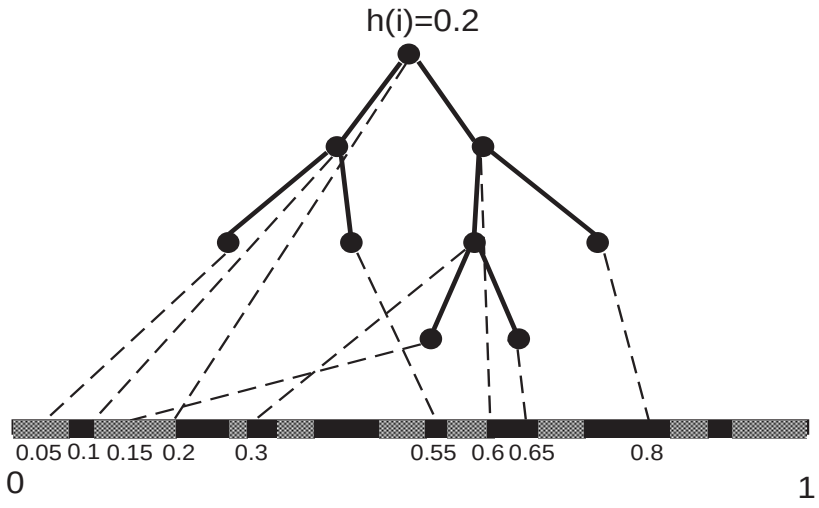


Figure 2.4: The mapping of an active tree to the servers. The bold edges represent the active tree. The dashed edges represent the mapping. The $[0,1)$ interval is divided to segments which represents the servers.

function h is randomly chosen from a family of hash functions $\mathcal{H}$ so that $h(i)$ is uniformly distributed in I. This property is sometimes called *one-wise independence*. We stress that this is a very weak requirement, (for instance the common notion of a pairwise independent family of hash functions satisfies this requirement). As before we need to count the number of active nodes covered by V. Denote by B_v the number of nodes covered by V. Now B_v depends both on the randomness of the routing algorithm and on the randomness of the hash function.

Lemma 2.4.7. *If $h(i)$ is uniformly distributed in I and $|s(V)|$ is $O(\frac{\log n}{n})$, then for every $t > 0$:*

$$\Pr[B_v \geq t] \text{ is } O\left(|s(V)| \cdot \frac{q}{c} \cdot 2^{-\Theta(t)}\right)$$

where the probability is over the choice of the hash function h and the routing algorithm.

Proof. Consider level j of the path tree. According to Observation 2.4.3, $s(V)$ covers at most $\lceil |s(V)|2^j \rceil$ active nodes. Lemma 2.4.4 bounds the depth of the active tree by $\log(q/c) + O(1)$, and $|s(V)|$ is bounded by $\log n$. We conclude V covers at most $O(1)$ nodes from each level of the active tree. Therefore if $B_v \geq t$ then V covers at least one node from the first $\log(q/c) - \Theta(t) + O(1)$ levels of the tree.

We proceed by calculating the probability V covers at least one node from level j. Every specific level j node of the path tree is uniformly distributed (by the hash function) in a segment of length 2^{-j}. For instance the leftmost node of level j is uniformly distributed in the segment $[0, 2^{-j}]$ and so on. Therefore the expected number of nodes V covers in level j is $O(|s(V)|2^j)$. Conclude that the probability V covers at least one node in level j is bounded by $O(|s(V)|2^j)$.

Now we have: $\Pr[B_v \geq t] \leq \Pr[V$ covers at least one point in the first $\log(q/c) - \Theta(t) + O(1)$ levels$]$ which is at most

$$\sum_{j=0}^{\log(q/c)-\Theta(t)+O(1)} |s(V)|2^j = O\left(|s(V)| \cdot \frac{q}{c} \cdot 2^{-\Theta(t)}\right).$$

□

By linearity of expectation, the expected number of active nodes V covers is $O(|s(V)|\frac{q}{c})$. When $|s(V)|$ is $\Theta(1/n)$ (the graph is smooth) and $q \leq n$ the bound translates to $O(1/c)$. Therefore the expected number of requests handled by V is $O(|s(V)|q)$, which is $O(1)$ when the graph is smooth.

2.4.4 Multiple Hotspots

In this Section we analyze the more general case where there is an arbitrary set of demands for n data items. We make the following assumptions: the threshold is set $c = \Theta(\log n)$, the demand for data item i is q_i such that $\sum_i q_i = n$ and that the hash function h which maps the data items into I is randomly chosen from a $\log n$-wise independent family.

The following is the main result of this Section:

Theorem 2.4.8. *If h is $k-$wise independent where $k \geq \log n$ then for all sequences of n requests:*

(i) - Given that $|s(V)| \leq \frac{\log n}{n}$ for each server V, with probability at least $1 - \frac{1}{n}$ the maximum number of items stored at any of the caches is $O(\log n)$, where the probability is taken over the random choices of the routing algorithm and the hash function.

(ii) - For each server V the expected number of times V supplies a data item is $O(|s(V)|n)$. Given that $|s(V)| \leq \frac{\log n}{n}$, with probability $1 - \frac{1}{n}$, each server supplies data items at most $O(\log^2 n)$ times.

Proof. Fix a server V. Denote by p_i the probability V supplies data item i. According to Lemma 2.4.7 we have that p_i is $O(\frac{q_i|s(V)|}{c})$. Denote by L_v the number of data items supplied by V. We have

$$\mathbb{E}[L_v] = \sum_i p_i = O\left(\sum_i \frac{q_i|s(V)|}{c}\right) = O\left(\frac{n|s(V)|}{c}\right).$$

Recall that $c \geq \log n$ and $|s(V)| \leq \frac{\log n}{n}$ so $\mathbb{E}[L_v]$ is $O(1)$. The random variable L_v is the sum of $\log n$-wise independent Bernoulli variables. Using Claim 2.2.15 we have that:

$$\Pr[L_v \geq \delta \log n] \leq \Pr\left[|L_v - \mathbb{E}[L_v]| \geq O(\delta \log n \cdot \mathbb{E}[L_v])\right] \leq O\left((\delta^2 \log n)^{\frac{1}{2}\log n}\right)$$

The proof of part (i) is completed by setting the constant δ large enough so that $\Pr[L_v \geq \delta \log n] \leq \frac{1}{n^2}$ and union bounding the probabilities for the n servers.

Part (ii) is composed of two statements. The first follows directly from Lemma 2.4.7 which states that the expected number of times V handles the i'th data item is $\mathbb{E}[q_i|s(V)|]$. Therefore the expected number of times V supplies a data item is $O(\sum_i q_i|s(V)|) = O(|s(V)|n)$.

For the second part let B_i denote the random variable counting the number of active nodes V covers from the i^{th} active tree. According to Lemma 2.4.7 there are some constants a, p such that $\Pr[B_i \geq t] \leq a \cdot p^t$ where a is $O(\frac{|s(V)|q_i}{c})$ and $p = \Theta(1)$. In other words B_i is stochastically dominated by a random variable of the form $X_i = aG_i$ where G_i is a geometric variable with a parameter which is $\Theta(1)$. So $\Pr[\sum B_i \leq O(\log n)] \leq \Pr[\sum X_i \leq O(\log n)]$. Furthermore w.l.o.g we may assume that $a < 1$ is small enough so that $Var[X_i] \leq \mathbb{E}[X_i]$, as a change in a may be incorporated into the O notation. $\mathbb{E}[\sum X_i]$ is $O(\sum \frac{|s(V)|q_i}{c})$ which is $O(1)$. Now Claim 2.2.15 implies that for ϵ large enough

$$\Pr\left[\sum X_i \geq 2\epsilon \log n\right] \leq \Pr\left[\left|\sum X_i - \mathbb{E}[\sum X_i]\right| \geq \epsilon \log n\right] \leq \left(\frac{\mathbb{E}[\sum X_i]}{\epsilon^2 \log^2 n}\right)^{\log n} \leq \frac{1}{n^2}$$

which concludes the proof of Theorem 2.4.8. □

Content Update: The tree like structure of the cache means that the popular data item may be changed or altered efficiently by propagating the update from the owner of the data (the root) along the active tree. As a consequence, an update of all the caches in which the data item is stored would take $O(\log \frac{q}{c}) \leq O(\log n)$ time and $O(\log \frac{q}{c}) \leq O(\log n)$ messages.

Summary We have shown that our caching scheme satisfies the properties required: Hot spots are eliminated with high probability for every set of requests, caches of servers are small, and most strikingly - the caching scheme causes *no extra delay* for obtaining the data item. All done in a dynamic and scalable manner. The caching scheme uses the fact that in the Distance halving *continuous* graph, every node is the root of an infinite binary tree. The same property was also used by Nadav and Naor [98] in order to build fault tolerant storage systems.

2.5 Emulating General Graphs - Smoothness is Everything

In this section we show how our technique can be used to dynamically construct a graph which embeds *any* family of fixed degree graphs. Theoretically speaking, this result implies that considering scalable systems separately is superfluous; i.e. any problem could be solved in a static environment and then be made dynamic via this technique. The main disadvantage of this technique is that the dependency on the smoothness is heavier, so tailored designs are indeed interesting. General techniques for constructing network topologies were independently suggested by Abraham *et al* [2].

Let $\{G_1, G_2, \ldots\}$ be an infinite family of regular graphs with degree d. Assume that G_i has 2^i vertices called $u_{i_1}, u_{i_2}, \ldots, u_{i_{2^i}}$. We show how a smooth set of n points $\vec{x}$ in $[0,1)$ can be used to construct a graph $G_{\vec{x}}$ that emulates $G_{\lceil \log n \rceil}$. Assume that $\vec{x}$ is smooth and that node V_i is associated with point x_i. Assume for now that all nodes know n (this assumption would be removed later). For each k, define the function Φ_k from the nodes of G_k to the nodes of $G_{\vec{x}}$ as follows:

$$\Phi_k(u_{k_j}) = V_i \quad \text{if} \quad \tfrac{j}{2^k} \in s(x_i).$$

The function Φ_k spreads the nodes of G_k evenly among the nodes of $G_{\vec{x}}$. Note that the function Φ_k could be computed locally, i.e. each node V_i can calculate which nodes in G_k are mapped to it.

Now if all the nodes of $G_{\vec{x}}$ can *agree on the same value of* k, then the edges of $G_{\vec{x}}$ are defined in the usual continuous-discrete style:

$$E(G_{\vec{x}}) = \left\{(V_i, V_j) \mid \exists (u_{k_i}, u_{k_j}) \in E(G_k)\ ,\ \Phi_k(u_{k_i}) = V_i, \Phi_k(u_{k_j}) = V_j\right\}.$$

In this case we say that edge (u_{k_i}, u_{k_j}) is *simulated* by (V_i, V_j). If all nodes in the network know what n is then all of them set $k = \lceil \log n \rceil$. It is routine to verify the following properties:

1. Every node in $G_{\vec{x}}$ simulates at most ρ nodes in G_k.
2. Every edge in $G_{\vec{x}}$ simulates at most ρ^2 edges in G_k.
3. The degree of $G_{\vec{x}}$ is at most $\rho \cdot d$.

In other words if $\vec{x}$ is smooth then $G_{\vec{x}}$ is a *real time emulation* of G_k. Particularly this means *any* computation performed by G_k, could be performed by $G_{\vec{x}}$ in constant slow down. see ([83], [71]) for an overview on the literature of real time emulations. It is important to notice that assuming all nodes know what n is, each node in $G_{\vec{x}}$ can calculate separately which are its neighbors. Next we remove the assumption that n is known. Smooth $\vec{x}$ implies that each V_i can calculate an estimation of n, denoted by n_i, by setting $n_i = \frac{1}{|s(V_i)|}$. By definition $\max_{i,j} \frac{n_i}{n_j} = \rho(\vec{x})$. Therefore it holds that $\forall i \quad \log n_i - \log \rho \leq \log n \leq \log n_i + \log \rho$. If $\vec{x}$ is guaranteed to have smoothness of at most ρ then each node can calculate a list of length $2 \log \rho$ that contains $\lceil \log n \rceil$. Each node V now would set edges according to every index in its list; i.e. establish connections resulting from the union of the Φ's on its list. The following Theorem is summarizes the Section:

Theorem 2.5.1. *When $G_{\vec{x}}$ is constructed as described it holds that*

1. *The degree of $G_{\vec{x}}$ is at most $2d \cdot \rho \log \rho$.*

2. *If ρ is a constant then the graph $G_{\vec{x}}$ can emulate in real time the graph $G_{\lceil \log n \rceil}$.*

2.6 Load Balancing the ID's - Achieving Smoothness

We have seen that the smoothness of the decomposition of I determines the efficiency of many of our protocols. In this section we suggest various distributed algorithms in which a node can choose its ID. (i.e. perform the first step of Algorithm Join in Section 2.2). The goal is that all nodes choose their ID such that I is smoothly divided between them. Algorithms similar to ours were independently suggested in [2] and [85]. Subsequent papers by Karger and Ruhl [64] and Manku [66] generalize these results.

A straightforward algorithm, that was also suggested by previous constructions (e.g. [120],[84]) is letting each node choose its ID by sampling randomly and uniformly a point in I:

Algorithm Single Choice for node V:

1. Choose $V.ID$ uniformly at random in $[0, 1)$.

The following lemma is proven in [84].

Lemma 2.6.1. *After inserting n random points the length of the longest segment is w.h.p $\Theta(\frac{\log n}{n})$. With high probability there is no segment which is shorter than $\Theta(\frac{1}{n^2})$*

An important property of the Single Choice Algorithm is that the distribution of the ID's remains unchanged when a random node is deleted from the network. Therefore Lemma 2.6.1 holds as long as nodes join and leave randomly. In the Single Choice Algorithm segments remain small w.h.p, and in fact a careful look at our proofs shows that this is enough, i.e. using this scheme would only cause a logarithmic blowup in parameters. The main drawback of the scheme is that it may allow some segments to be very small - of length $O(\frac{1}{n^2})$, which means that the nodes covering them would hardly share any of the load. A slight improvement is the following.

Improved Single Choice Algorithm for node V_i:

1. Choose a point $z \in [0,1)$ uniformly at random.
2. Lookup z and find the boundaries of the segment of the node which currently covers z.
3. Set $V.ID$ to be the *middle* point in that segment.

Lemma 2.6.2. *In the improved single join algorithm, the shortest segment would be of length* $\Theta(\frac{1}{n \log n})$ *w.h.p and the longest segment would remain* $O(\frac{\log n}{n})$.

Proof. Simulate the process of choosing the $x_1, x_2, \ldots, x_n$ ID's by growing a random binary tree in the following way: Let $z_1, z_2, \ldots, z_n$ denote the n random points chosen at Step (1) of the algorithm, where z_i is the point chosen by the ith node joining. The point x_1 always takes the value $\frac{1}{2}$, which corresponds to the root of the tree. The point x_2 takes a value of $\frac{1}{4}$ with probability $\frac{1}{2}$ (if $0 \leq z_2 < \frac{1}{2}$), in which case we add a left child to the root. The point x_2 takes a value of $\frac{3}{4}$ with probability $\frac{1}{2}$, in which case we add a right child to the root, and so on. Each time a point is added, it randomly selects a path from the root until it reaches a leaf (the path is determined by the binary representation of z), and then becomes either the left or the right child of that leaf. Now consider the layer of the tree in which there are $\Theta(n \log n)$ nodes. For each node of the tree, with high probability there would be at most one of z-points which reached it (balls and bins). It immediately follows that the smallest segment would be of length $\Omega(\frac{1}{n \log n})$ with high probability. Consider the layer of the tree which has $\Theta(n/\log n)$ nodes. With high probability each one of them were hit by at least one of the z-points. It follows that the largest segment is of size $O(\frac{\log n}{n})$. □

The idea of the following algorithm (following the spirit of the two choice paradigm, see [96] for a survey), is to let a joining node choose *many* locations and set its ID to be the best location found.

Multiple Choice Algorithm:

1. Estimate $\log n$.
2. Sample $t \log n$ random points from I, when t is some constant to be determined later.
3. Check all segments containing those points. Let s be the longest of these segments. Set ID to be the middle point of s.

Estimating $\log n$ is a simple task, see Section 2.3 for details. For convenience we assume that the estimation is completely accurate. The problems caused by the inaccuracy of the estimation could be overcome by slightly enlarging the parameter t.

Lemma 2.6.3. *If* $t \geq 2$, *after* n *points are inserted, the length of the shortest segment is at least* $\frac{1}{4n}$ *with probability* $1 - \frac{1}{n}$.

Proof. A segment of length $\frac{1}{4n}$ could be created only if all the $t \log n$ samples fell in segment of length at most $\frac{1}{2n}$. The probability a random point in I falls in some a segment of length $\frac{1}{2n}$ is at most $\frac{1}{2}$. The probability all the $t \log n$ fell in these segment is at most $2^{-t \log n} = \frac{1}{n^t}$. Now, since $t \geq 2$ a union bound over the error probabilities of all the points proves the Lemma. □

In the following we show that even when an adversary controlled the initial state of the system, after the injection of n more points, with high probability no segment would be big.

Theorem 2.6.4. *For any $m \geq n$ and **any** configuration of m points in I, after inserting n points, the largest segment would be of size at most $\frac{2}{n}$, with probability $1 - \frac{1}{n}$.*

Proof. First we prove the following lemma:

Lemma 2.6.5. *Assume the longest segment is of length $\frac{c}{n}$ (c may be a function of n), then for sufficiently large t, after inserting $\frac{2n}{c}$ points, the longest segment would be of length at most $\frac{c}{2n}$ with probability $1 - \frac{1}{n^2}$.*

Proof of Lemma 2.6.5. Let s be a segment such that $|s| = \frac{c}{n}$, i.e. s is a segment of maximum length. In step (1) of the algorithm $t \log n$ random points are chosen. Whenever one of these points is in s we say that s is *hit*. Let A be a random variable that counts the number of times s was hit after $\frac{2n}{c}$ points were inserted. We have

$$\mathbb{E}(A) = \frac{c}{n} \cdot \frac{2n}{c} \cdot t \log n = 2t \log n$$

By Chernoff's inequality

$$\Pr\left[A \leq (1-\delta)\mathbb{E}(A)\right] \leq n^{-\delta^2 t}.$$

If parameters are chosen such that $\delta^2 t \geq 3$ then by the union bound, with probability $1 - \frac{1}{n^2}$ *all* segment of length $\frac{c}{n}$ were hit at least $(1-\delta)2t \log n$ times. Now assume that there are k segments of size $\frac{c}{n}$. The total number of hits in all these k segments is at least $k(1-\delta)2t \log n$, with probability $1 - \frac{1}{n}$. Each inserted point creates at most $t \log n$ hits, therefore there were at least $2k(1-\delta)$ points in which hit a large segment. We have that if $2(1-\delta) \geq 1$ then all segments of length $\frac{c}{n}$ were split. Both conditions hold when $\delta = \frac{1}{2}$ and $t = 12$. □

The proof of Theorem 2.6.4 is completed by applying the previous Lemma iteratively $\log n - 1$ times, thus inserting n new points and making sure that all segments are of length at most $\frac{2}{n}$. □

2.6.1 Handling Deletions

The Multiple Choice Algorithm achieves a smooth set of points in the pure Join model - when node may join the system but not leave it. Achieving smoothness in a setting which allows deletions requires a more complicated scheme. We will show two schemes, the first handles random deletions only, the second deals with adversarial deletions as well.

The most naive way to handle the deletion of a point is to assign its segment to its predecessor on the ring. It is easy to see that this naive algorithm does not suffice: Assume that there are $2n$ points spread smoothly in I, and randomly delete each one with probability $\frac{1}{2}$. With high probability there would be a sequence of $\Omega(\log n)$ *consecutive* points that were deleted, creating a segment of length $\Omega(\frac{\log n}{n})$ and violating the smoothness. We conclude that some balancing mechanism must be implemented in order to maintain smoothness under deletions.

The Bucket Solution The 'bucket' solution was suggested in Viceroy [84]. In Viceroy the nodes join the system with the Single Choice algorithm, and the *bucket scheme* is used balance the nodes both when nodes join and leave. The balance is achieved via an extra structure. We maintain a distributed coordination mechanism between contiguous chains of nodes, consisting of $O(\log n)$ nodes each. We call such a group of $O(\log n)$ nodes a *bucket*. Inside each bucked we maintain a simple ring (which mostly overlaps the larger ring of the DH construction). The buckets are maintained such that two properties hold:

1. The size of the bucket is always $\Theta(\log n)$. When the size of a bucket exceeds $c \log n$ (for some constant c) it splits into two. When the size of a bucket shrinks bellow a threshold, it merges with a neighboring bucket. An estimation of $\log n$ is maintained within each bucket.

2. Within each bucket segments are distributed evenly, i.e. nodes may change their ID *slightly* so that no segment would be too big or too small.

The exact way in which Step (2) is performed may vary according to context. Keeping all segments within a bucket of equal length at all times may have a large overhead, as all members of the bucket update their state whenever a node join or leaves. Therefore it makes more sense for the nodes in the bucket to rearrange themselves only when the smoothness within the bucket exceeds some parameter.

The correctness of the bucket solution follows from the fact that w.h.p every interval of length $\frac{\log n}{n}$ would contain $\Theta(\log n)$ points (balls and bins). Thus, when a segment becomes too big or too small a balancing within the bucket suffices.

The Cyclic Scheme In the following we sketch a deterministic scheme which guarantees smoothness even when an adversary determines when and which nodes join and leave the system. It takes $\Theta(\log n)$ messages to perform both Join and Leave, and it requires an additional structure. The idea is to divide the segments into long segments and short segment, and to have all the long segments in one *contiguous* interval whose starting point is marked by the point a and its ending by point b. When a node wishes to enter the system, it does so by splitting into two the long segment adjacent to b (see Figure 2.5). When a node wishes to leave the system, it swaps positions with the node in charge of the *first* short segment (whose ID. in Figure 2.5 is a) and then leaves creating a long segment. If a node wishes to leave the system and there is only one short segment in the system. then a slightly more delicate operation should be done. The leaving node should swap places with the node covering the short segment and then leave. The resulting decomposition has short segments, long segments and at most one medium sized segment whose length is larger than the short ones but smaller than the long ones. Thus the cyclic scheme guarantees a smoothness of 2 even under adversarial behavior.

The main difficulty is finding quickly the node at the beginning or the end of the interval of short segments. This could be done by maintaining an extra structure of two $2-3$ trees on top of the DHT, in which points a and b are roots. A $2-3$ tree guarantees that points a and b could be found in $O(\log n)$ time. Furthermore the insertion and deletion of nodes would only cause an overhead of $O(\log n)$ messages and updates for maintaining the tree. One difficulty is that nodes which are close to the roots of the trees suffer from high load in the sense that they handle almost every joining and leaving. This could be dealt with by rotating each edge once in a while, i.e. once in a while an inner node randomly chooses one of its children and swaps positions with it, thus over

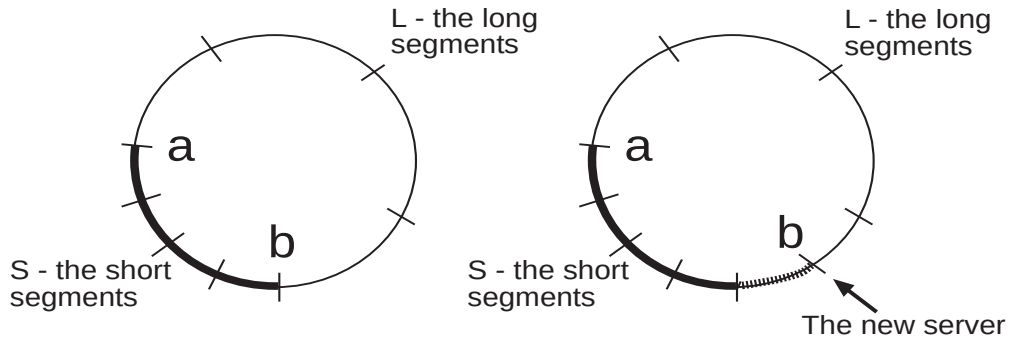


Figure 2.5: In the cyclic scheme, an entering node splits the first long segment.

time all nodes spend most of the time as leaves of the tree. Note that this swap is in the $2-3$ tree and not in the DHT so it does not involve all the overhead of changing location in the DHT itself.

Theoretically speaking the cyclic scheme is appealing, as it offers excellent smoothness in the worst case even under adversarial insertions and deletions. Its main drawback is its vulnerability to concurrent joining and leaving. Say m nodes wish to leave the system in the same time. It is not clear how to coordinate them such that the smoothness property remains. A locking mechanism that would ensure that nodes leave (and join) one at a time implies a low throughput of the system and seriously limits its dynamic nature. Thus it seems that the cyclic scheme is less practical than the previous schemes discussed in this Section.

2.7 Higher Dimensions

So far we limited ourselves to the case where I is a one dimensional unit interval. Indeed, as was shown by previous constructions a one dimensional name space suffices for every topology, and is also good for a tailored design of Distributed Hash Tables. In some applications it turns out to be useful to apply the same approach to a two dimensional universe. In Chapter 3 the continuous-discrete approach is used to construct dynamic *quorum system*. In this Section we show how to construct a network which is an expander.

In the two dimensional case the set of points cannot be associated with segments, but rather be associated with a tessellation of the plain into cells, such that each point is associated with a cell and is responsible for all keys that fall within that cell. This was done in CAN [113] where the plain was divided into rectangles. We present a simpler way to do it using the points as generators to a planar ordinary Voronoi diagram.

2.7.1 Dynamic Voronoi Diagrams

Definition 2.7.1 (planar ordinary Voronoi diagram). Given a finite number (at least 2) of distinct points in the Euclidean plane, we associate all locations in that space with the closest member(s) of the point set with respect to the Euclidean distance. The result is a tessellation of the plane into a set of regions associated with members of the point set. We call this tessellation the *planar ordinary Voronoi diagram* generated by the point set, the points are sometimes referred to as

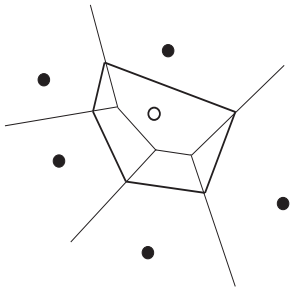


Figure 2.6: Addition of a new generator to a Voronoi diagram.

generators and the regions constituting the Voronoi diagram *Voronoi cells*. The dual triangulated graph is called the *Delaunay triangulation*. See Okabe *et al* [107] for a thorough overview of Voronoi diagrams and their applications.

See Okabe *et al* [107] for a thorough overview of Voronoi diagrams and their applications. Given an existing Voronoi diagram, the entrance of a new generator and the exit of an existing one affects only the cells adjacent to the location of the generator. Therefore a Voronoi diagram can be maintained by a distributed algorithm in which every cell is calculated separately and *locally*. The time and memory needed to compute a single Voronoi cell is $\Theta(d)$ when d is the number of neighbors the cell has; i.e., the degree of the generator in the Delaunay tessellation. See Figure 2.6 for a demonstration of an insertion of a new generator. It is known that d is always 6 on average (Euler's formula), but might be as high as $n-1$.

In the following we set $I = [0,1) \times [0,1)$. Let $\vec{x}$ be a set of n points in I.

Definition 2.7.2. We say that $\vec{x}$ has *smoothness* ρ if the following two conditions hold: (1) when dividing the rectangle to ρn rectangles of size $\frac{1}{\sqrt{\rho n}} \times \frac{1}{\sqrt{\rho n}}$, each rectangle contains at least one point from $\vec{x}$. (2) when dividing the rectangle to $\frac{n}{\rho}$ rectangles of size $\sqrt{\frac{\rho}{n}} \times \sqrt{\frac{\rho}{n}}$, each rectangle contains at most one point from $\vec{x}$.

The Join/Leave operations

Nodes are associated with generators of a Voronoi diagram. Each node holds its own location on the plane and the location of its neighbors in the Delaunay triangulation. A node that wishes to join the system does the following:

Single Choice Algorithm:

1. Choose a location x in the unit square (typically x would be chosen randomly and uniformly from $[0,1) \times [0,1)$).
2. Find the node whose cell contains x. Learn the location of its neighbors.
3. Set x as a new generator of the diagram. Calculate the boundaries of the new Voronoi cell and inform the neighbors so that they can update their tables.

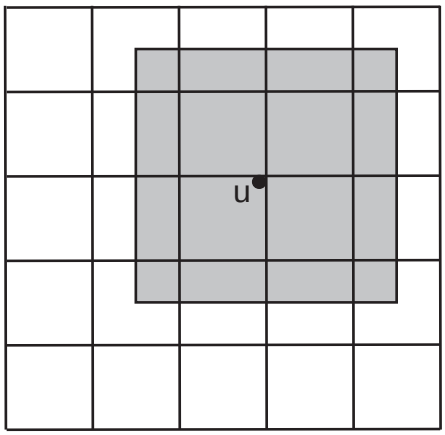


Figure 2.7: If each square contains at least one Voronoi generator, then the cell generated by u is contained in the shaded area.

Before analyzing the algorithm we show the properties of a Voronoi diagram in which the location of each generator was chosen randomly and uniformly. We show that with high probably the Voronoi diagram decomposes the square into more or less equal cells.

Theorem 2.7.3. *If the location of each generator of the Voronoi diagram was chosen uniformly and randomly in* $[0,1) \times [0,1)$ *then with high probability the following holds:*

1. *The area of the largest Voronoi cell is* $O(\frac{\log n}{n})$.
2. *The number of neighbors each Voronoi cell (the maximum degree of the Delaunay graph) is* $O(\log n)$.
3. *The projection of each Voronoi cell on the axis lines is at most* $O(\sqrt{\log n/n})$.

Proof. Divide the square into $\frac{n}{\log n}$ squares, each of size $\sqrt{\frac{\log n}{n}} \times \sqrt{\frac{\log n}{n}}$. Now model the process as putting n balls in $\frac{n}{\log n}$ bins. It is well known that when n balls are put uniformly at random into $\frac{n}{\log n}$ bins, then w.h.p every bin contains $\Theta(\log n)$ balls. Assume this high probability event occurs and each small square contains $\Theta(\log n)$ balls. Fix a generator x_i. A simple geometric argument demonstrated in Figure 2.7.1 shows that all the neighbors of x_i must lie within the 25 squares that compose the 5×5 grid which surrounds the square of x_i. This asserts claims $(1), (3)$. Since each square contains $O(\log n)$ generators the number of neighbors of x_i is also bounded by $O(\log n)$. □

Since the computation of a Voronoi cell is a local operation, Step (3) of the Join algorithm takes $O(d)$ time and memory, where d is the degree of the Voronoi cell in the Delaunay graph. The average degree is 6 and Theorem 2.7.3 assures that w.h.p all degrees are at most $O(\log n)$. Step (2) of the the algorithms requires locating the node whose cell contains the point x. The complexity of Step (2) depends upon the topology of the network and the search options it provides. If the topology of the network is that of the Delaunay graph, then the node holding x could be found by a greedy algorithm along the geometry of the Voronoi diagram; i.e., the query moves along the Delaunay edges in a greedy way to the direction of x. Thus the time complexity and the message complexity of Step (2) are $O(\sqrt{n})$. A similar approach is taken in CAN [113]. Additional structure of the network may reduce the complexity of Step (2). The Distance Halving DHT (or any other efficient DHT) may be used to perform Step (2) in $O(\log n)$ time and $O(\log n)$ messages.

The Leave operation is done similarly. When a nodes wishes to leave the system, it informs its neighbors which in turn divide and redistribute the area of its cell among themselves.

2.7.2 Achieving Smoothness in Two Dimensions

In this section we show that a natural generalization of the Multiple Choice algorithm achieves smoothness with high probability in the two dimension setting. We need to assume that the network supports a lookup operation for the points in I (for instance by a DHT). For a point $z \in I$ define $r(z)$ to be the rectangle containing z when I is divided to $2n$ rectangle of size $\frac{1}{\sqrt{2n}} \times \frac{1}{\sqrt{2n}}$. Let $R(z)$ be the rectangle containing z when I is divided to $n/2$ rectangle of size $\sqrt{\frac{2}{n}} \times \sqrt{\frac{2}{n}}$.

2D Multiple Choice

1. Estimate n.
2. Sample $t \log n$ random points in I. Call these points $z_1, z_2, \ldots, z_{3 \log n}$.
3. For each z_i perform Lookup(z_i) and check whether $r(z_i)$ and $R(z_i)$ are empty.
4. If there exists an i such that $r(z_i)$ is empty and $R(z_i)$ is empty then set $x \leftarrow z_i$.
5. Otherwise find an i such that $r(z_i)$ is empty and set $x \leftarrow z_i$. If no such i exists then we say the algorithm failed and set $x \leftarrow z_1$.

We assume for convenience that the estimation of n is accurate. A multiplicative estimation of n is easily achievable and suffices.

Lemma 2.7.4. *After inserting n points using the 2D Multiple Choice algorithm, with probability $1 - \frac{1}{n}$ it holds that the smoothness of $\vec{x}$ is as most 2.*

Proof. First we show that w.h.p probability every small rectangle will contain at most one point from $\vec{x}$, i.e. Step (4) of the algorithm never fails. There are $2n$ small rectangles, at most n of them are not empty. The probability to hit a non empty rectangle is at most $\frac{1}{2}$. The probability that all $t \log n$ samples hit a non empty rectangle is at most $\frac{1}{n^t}$. Thus, the probability the algorithm failed for some point is at most $\frac{1}{n^2}$ for any $t \geq 3$.

Next we need to show that at the end of the algorithm w.h.p all the big rectangles contain at least one point. There are $n/2$ big rectangles. Assume k of them already contain a point from x. The probability a random point from I hits one of these k rectangles is $\frac{2k}{n}$. The probability that when inserting α_k points only non-empty rectangles were hit is at most $\frac{2k}{n}^{\alpha_k t \log n}$. This implies that when $\frac{3}{t \log(n/2k)}$ points are inserted, the probability of hitting non-empty rectangles only is at most $\frac{1}{n^3}$. Now for some $t = O(1)$ it holds that

$$\sum_{k=0}^{n/2-1} \frac{3}{t \log(n/2i)} \leq \frac{3}{4} n$$

which means that with probability $1 - \frac{1}{n^2}$ after inserting $\frac{3}{4}n$ points, all the big rectangles are full. The proof is completed by union bounding the error probabilities. □

2.7.3 Constructing Expanders

Here we see how a two dimensional name space could be used to create expanders in a P2P setting. Expander graphs are graphs that are very 'well connected' in the sense that for every set of vertices S of size at most $\frac{1}{2}|V|$ there are at least $\alpha|S|$ vertices in $V \setminus S$ that are adjacent to some vertex in S. In this case we say the *expansion* of the graph is α. Expander graphs are probably one of the most researched structures in combinatorics. They have numerous applications in computer science. Applications in distributed computing include load balancing, fault tolerance and search through random walks (c.f. [7, 36, 76, 40, 48, 17]).

It is well known that a random regular graph is an expander with high probability [43]. Independently from this paper, Law and Siu [73] used this fact to construct an expander (w.h.p) in a P2P setting. An explicit and deterministic construction for expanders was given by Margulis [89] and Gabber and Galil [44], and was later generalized by Cai [27]. We use the continuous-discrete approach in order to construct this expander in a P2P setting. Gabber and Galil define a continuous graph G over I by the following two transformations: $f(x,y) = (x+y, y) \mod 1$, $g(x,y) = (x, x+y) \mod 1$. The neighbors of point $(x,y) \in I$ are $f(x,y), g(x,y), f^{-1}(x,y), g^{-1}(x,y)$. For any set $A \subseteq I$ define $\delta(A)$ to be the set of points in $I \setminus A$ which are neighbors of a point in A. Denote by $\mu(A)$ the area of the set A (its Lebesgue measure).

Theorem 2.7.5 ([44]). *For every set A of points in I such that $\mu(A) \leq \frac{1}{2}$ is well defined, it holds that $\mu(\delta(A)) \geq \frac{(2-\sqrt{3})}{2}\mu(A)$.*

Corollary 2.7.6. *Let $\vec{x}$ be a set of n points in I. Let $G_{\vec{x}}$ be the discretization of the Gabber-Galil continuous graph. The maximum degree of $G_{\vec{x}}$ is $\Theta(\rho)$, and the expansion of $G_{\vec{x}}$ is $\Omega(\frac{(2-\sqrt{3})}{\rho})$. So if ρ is constant $G_{\vec{x}}$ is a constant degree expander.*

Any network created by maintaining a Voronoi diagram of a smooth set of points, would guarantee expansion. Currently no know routing scheme is known for the general Gabber-Galil expander. A step towards a routing scheme was given by Larsen [72] which provides an algorithm for finding short routes in the Gabber-Galil expander whenever the number of nodes is a square of a prime. In this case the construction yields an efficient constant degree expanding DHT. In contrast, in the construction of Law and Siu [73], in order for the expander to be navigable the degree should be logarithmic.

Chapter 3

Scalable and Dynamic Quorum Systems

Summary: We investigate issues related to the probe complexity of quorum systems and their implementation in a dynamic environment. Our contribution is twofold. The first regards the algorithmic complexity of finding a quorum in case of random failures. We show a tradeoff between the load of a quorum system and its probe complexity for non adaptive algorithms. We analyze the algorithmic probe complexity of the *Paths* quorum system suggested by Naor and Wool in [105], and present two optimal algorithms. The first is a non adaptive algorithm that matches our lower bound. The second is an adaptive algorithm with a probe complexity that is linear in the cardinality of the smallest quorum set. We supply a constant degree network in which these algorithms could be executed efficiently. Thus the *Paths* quorum system is shown to have good balance between many measures of quality. Our second contribution is presenting *Dynamic Paths* - a suggestion for a dynamic and scalable quorum system, which can operate in an environment where elements join and leave the system. The quorum system could be viewed as a continuous-discrete dynamic adaptation of the *Paths* system, and therefore has low load high availability and good probe complexity. We show that it scales gracefully as the number of elements grows.

3.1 Introduction and Motivation

Quorum systems serve as a basic tool providing a uniform and reliable way to achieve coordination between nodes in a distributed system. Quorum systems are defined as follows:

Definition 3.1.1. Let U be a universe of n elements. A set system $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ is said to be a *quorum system* over the universe U if $\forall i\ S_i \subseteq U$ and $\forall i, j\ S_i \cap S_j \neq \varnothing$. Each set S_i is referred to as a *quorum set* or simply as a *quorum*.

Quorum systems have been used in the study of distributed control and management problems such as mutual exclusion (cf. [46],[117]), data replication protocols (cf. [46]) and secure access control ([104]). In many applications of quorum systems the underlying universe is associated with a network of nodes, and a quorum is employed by accessing each of its elements. For example, in a typical implementation of mutual exclusion using quorum systems, processes request access to the critical section from all members of a quorum. A process can enter its critical section only if it receives permission from all nodes in a quorum. The intersection property guarantees the integrity

of the mutual inclusion. In a typical application of data replication, the quorum sets are divided into reading quorums and writing quorums where each reading quorum intersects each writing quorum. When a data item is added to the system, it is written into all the members of a writing quorum. A data item is searched by querying all the members of a reading quorum. The intersection property guarantees the effectiveness of the search. We investigate two aspects of quorum systems:

1. It is often assumed that nodes can somehow find and communicate with one another. We analyze algorithms for finding quorum systems in a distributed network while taking into account the *network implementation*; i.e., the network and the quorum system should be compatible such that elements from the same quorum are connected to one another. We supply algorithms for finding a quorum set (even in the case of failures) and analyze their running time and communication complexity. In this setting non-adaptive algorithms are attractive since they can be executed in parallel.

2. The setting in which the quorum operates is often dynamic, and should accommodate changes in the quorum system over time. See for instance [80],[117]. We address the problem of designing a quorum system that is fit for a scalable and dynamic environment where nodes leave and join at will. Abraham and Malkhi [3] address this problems when the intersection property is not guaranteed but rather occurs with high probability.

We suggest quorum systems that operate in a dynamic peer-to-peer model. We apply the continuous-discrete approach to an appropriate quorum systems, and provide the distributed algorithms for finding the quorums. We allow two types of events:

1. A Node may temporarily fail (halt). The failure of a node occurs with some fixed probability and is independent from failures of other nodes in the network. It is desired that the probability that a live quorum is found be as high as possible.

2. Nodes may wish to join the system or to leave it (a long term failure of a node could be regarded as if the node left the system). It is desired that the quorum sets be updated such that these nodes are included/excluded from the system.

3.1.1 Measures of Quality

The metrics that measure the quality of a dynamic quorum system relate both to its *combinatorial* structure and to its effectiveness when implemented in a distributed network. The following metrics were analyzed by Naor and Wool in [105] and are used to measure the quality of static systems as well.

- `Load` - A strategy is a distribution over quorum sets, giving each quorum set an access probability (i.e., the probability by which it is accessed by the user). A strategy induces a load on each element, which is the sum of the probabilities of quorums it belongs to. This represents the fraction of the time an element is used. For a given quorum system $\mathcal{S}$, the load $\zeta(\mathcal{S})$ is the minimal load on the busiest element, minimizing over the strategies. The load measures the quality of the quorum system in the following sense: if the load is low, then each element is accessed rarely, thus it is free to perform other unrelated tasks. Let c be the cardinality of the smallest quorum set. Naor and Wool prove in [105] the following lemma:

Lemma 3.1.2. *The load of a quorum system is always at least* $\max\{\frac{1}{c}, \frac{c}{n}\}$ *which implies that* $\zeta(\mathcal{S}) \geq \frac{1}{\sqrt{n}}$.

- `Availability -` Assuming that each element fails with probability p, what is the probability F_p, that the surviving elements do not contain any quorum? This failure probability measures how resilient the system is, and we would like F_p to be as small as possible.

The Load is especially important if the application of the quorum system involves replication of data, as was described in the previous section. In this case the load is proportional to the fraction of data each element has to hold, and therefore smaller load means that each node needs to allocate a smaller amount of memory. The notion of availability is important when dealing with *temporary faults*. The most common strategy to deal with faults is to *bypass* them; i.e., find a quorum set for which all nodes are alive. This introduces the following notion:

- `Algorithmic probe complexity` - The complexity of the algorithms for finding a quorum should be low. Even if all nodes are alive the *network* should allow easy access to elements of the same quorum system. In case some elements fail, finding a live quorum set can be a difficult algorithmic task. Peleg and Wool analyzed in [108] the probe complexity of several quorum systems. They assume that an adversary decides which elements fail and analyzed the number of elements needed to be probed before either a living quorum is found or an evidence for the lack thereof. They assume that each probing takes $O(1)$; i.e., they ignore the complexity caused by the implementation of the network. Hassin and Peleg extend these results in [58] to the case where each node fails with some fixed probability . The *Algorithmic probe complexity* is the actual time and message complexity needed to find a live quorum. It is determined by the *network* and by the quorum system. A related term is the *Cost of Failures* introduced by Bazzi [20]. Given a network implementation and an algorithm for finding quorums, the cost for failures measures the average communication overhead caused by encountering a faulty node.

The dynamic setting introduces another set of demands:

- `Integrity-` Typically a the joining of a new node or the departure of an existing one result in a modification of the quorum sets. The integrity of the system should be preserved in two aspects: First the intersection property must hold. Bearden and Bianchini suggest in [21] a protocol for an online adjustment of quorum systems without compromising the integrity of the intersection property *during* the adaptation. It is necessary that the adaptations themselves do not corrupt the intersection property of the quorum system; i.e., that the intersection property holds after the adaptations took place. The second aspect is application oriented. Quorums that were used in the past (say for mutual exclusion) might not be legal quorum sets after the adaptation. It is necessary that when an adaptation occurs, the intersection guarantee that the quorum system supplies the application is not compromised.

- `Scalability-` The number of elements in the quorum system may increase over time. The increase in the size of the system should maintain the good qualities of it, i.e., it should decrease the load on each node and increase the availability of the system. It is important that when the system scales the algorithmic probe complexity would remain low. Finally the Join and Leave operation should be applied with low time and message complexity.

3.1.2 Our Contributions

Our contributions are divided into two parts. In the first part, we show a tradeoff between the load and the non-adaptive probe complexity of quorum systems in the face of failures (Section 3.2). Our Theorem gives lower bound for non-adaptive probe complexity as a function of the load. In Section 3.3 we show a non-adaptive algorithm for finding a quorum in the *Paths* quorum system which is tight in that respect. We further show an adaptive algorithm for *Paths* with probe complexity $O(\sqrt{n})$, which is optimal (up to constants). Thus combined with the results in [105] the *Paths* system is the first quorum system shown to have an excellent balance between many somewhat contradictory measures of quality. In Section 3.4) we present our second main the contribution - the presentation and analysis of *Dynamic Paths*, a construction for a dynamic and scalable quorum system which could be viewed as a dynamic adaptation of the *Paths* system. To the best of our knowledge *Dynamic Paths* is the first scalable quorum system which is shown to have low load, high availability and good probe complexity. Thus it is an excellent candidate for an implementation of quorums in a dynamic distributed network.

3.2 Non Adaptive Algorithms vs. Load

A non adaptive algorithm for finding a live quorum is an algorithm which decides which elements to probe *before* it gains any knowledge regarding which elements failed and which did not. Non-adaptive algorithms are important in the context of a distributed network since they are easy to implement in parallel. It might be worthwhile to 'pay' in a higher message complexity, and reduce the total time complexity of the algorithm. As an illustrative example consider a quorum system in which only $\sqrt{n}$ elements participate in quorum sets. Clearly querying only those $\sqrt{n}$ elements is sufficient to find a live quorum. The drawback of this approach is that the load on these elements would be high (Lemma 3.1.2 implies that it would be at least $n^{-\frac{1}{4}}$). In this section we show a tradeoff between the load of a quorum system and its probe complexity for non adaptive algorithms.

Theorem 3.2.1. *Let $\mathcal{S}$ be a quorum system over universe U with a load of $\zeta = \zeta(\mathcal{S})$. Assume that each element in U fails with some fixed probability $p < \frac{1}{2}$. Let $X \subseteq U$ be a predefined set of elements such that*

$$\Pr[X \text{ contains a live quorum}] \geq \frac{1}{2} ,$$

then

$$|X| \geq \frac{1}{2\log(1/p)+1} \cdot \frac{\log(1/4\zeta)}{\zeta}.$$

In particular if $\zeta(\mathcal{S})$ is $O(\frac{1}{\sqrt{n}})$ then, $|X|$ is $\Omega(\sqrt{n}\log n)$.

Proof. Let $\mathcal{S}_{\mathcal{X}}$ be all the quorum sets contained in X, i.e., $\mathcal{S}_{\mathcal{X}} = \{S | S \in \mathcal{S} \wedge S \subseteq X\}$. Let $\mathcal{R}$ be all the sets which are an intersection of X with a quorum; i.e.,

$$\mathcal{R} = \{R | R = S \cap X, S \in \mathcal{S}\}.$$

By the intersection property each set $R \in \mathcal{R}$ intersects all the sets in $\mathcal{S}_{\mathcal{X}}$. Therefore, if for a set $R \in \mathcal{R}$ *all* elements in R fail then X does not contain a live quorum. We show that $\mathcal{R}$ must contain many *disjoint* sets of small cardinality. Let f be a distribution over quorum sets which imposes

the optimal load ζ, and let $a = |X|\zeta$. Distribution f induces a marginal distribution over the sets $R \in \mathcal{R}$ by taking $S \cap X$ for each sampled set S. Under this distribution, the expected size of R is at most a (i.e., $\mathbb{E}_f[|R|] \leq a$), otherwise the load on the elements of X would be higher than ζ. By Markov's inequality we have that with probability at least $\frac{1}{2}$ the sampled set is of size at most $2a$, so we have

$$\sum_{R:|R|\leq 2a} \Pr_f[R \text{ is sampled}] \geq \frac{1}{2} \tag{3.1}$$

On the other hand since the load induced by f is at most ζ we have

$$\forall x \in X \quad \sum_{R:x\in R} \Pr_f[R \text{ is sampled}] \leq \zeta \tag{3.2}$$

Next we show that inequalities (3.1) and (3.2) imply that $\mathcal{R}$ contains a collections $\mathcal{R}'$ of at least $\frac{1}{2\zeta}$ disjoint sets of size at most $2a$. To see this employ the following procedure: pick a set $Q \in \mathcal{R}$ such that $|Q| \leq 2a$ and put Q in $\mathcal{R}'$. Define $\mathcal{R}_Q \subset \mathcal{R}$ to be all the small sets in $\mathcal{R}$ which intersect Q, i.e., $\mathcal{R}_Q = \{R \in \mathcal{R} | R \cap Q \neq \varnothing \wedge |R| \leq 2a\}$. Now since Q has at most $2a$ elements then by inequality (3.2) we have

$$\sum_{R\in\mathcal{R}_Q} \Pr_f[R \text{ is sampled}] \leq 2a\zeta \tag{3.3}$$

Remove the sets $\mathcal{R}_Q$ from $\mathcal{R}$ and repeat the procedure by picking another set Q, until all the sets of cardinality $\leq 2a$ were removed. By inequalities (3.1) and (3.3) we can perform this procedure $\frac{1}{4a\zeta}$ times. Clearly all the sets Q chosen in this process are disjoint and of small cardinality. For each set $Q \in \mathcal{R}'$ the probability that all its elements fail is at least p^{2a}. Since the sets are disjoint, these events are mutually independent. In order for the probability of finding a live quorum to be at least $\frac{1}{2}$ we must have then that

$$\begin{aligned}
&(1-p^{2a})^{\frac{1}{4a\zeta}} \geq \frac{1}{2} \\
&\exp\left(-\frac{p^{2a}}{4a\zeta}\right) \geq e^{-1} \\
&2a \cdot \log(1/p) + \log a \geq \log(1/4\zeta) \\
&a \geq \frac{\log(1/4\zeta)}{2\log(1/p)+1}
\end{aligned}$$

Now since $|X| = \frac{a}{\zeta}$ this implies the theorem. □

Theorem 3.2.1 lower bounds the probe complexity of *non-adaptive* algorithms. The smaller the load is, the larger the probe complexity is. The *Paths* system has a load of $\Theta(\frac{1}{\sqrt{n}})$. The theorem implies that any non-adaptive algorithm would have to probe a predefined set of $\Theta(\sqrt{n}\log n)$ nodes, in order to succeed with probability $\frac{1}{2}$. If the load is very large, say constant, then the bound given by Theorem 3.2.1 is $\Omega(1)$. In case of the Majority system, the bound is much worst than the trivial lower bound of $\frac{n}{2}$. This however is unavoidable since quorum systems with high load may have quorum sets of small cardinality, so any bound which uses the load alone will deteriorate when the load increases.

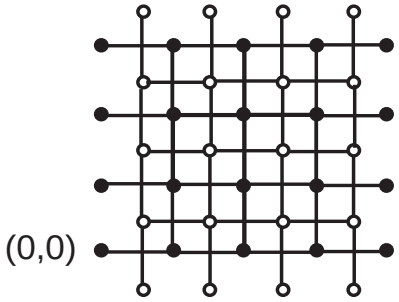


Figure 3.1: The *Paths* quorum system. The grids $G(3)$ (thick lines) and $G^*(3)$ (thin lines).

3.3 The Paths Quorum System

We recall the construction of the *Paths* system from [105]. We start with a precise definition of the grid we will be using.

Definition 3.3.1. Let $G(\ell)$ be the subgrid of $\mathbb{Z}^2$ with vertex set $\{(v_1, v_2) \in \mathbb{Z}^2 : 0 \leq v_1 \leq \ell+1, 0 \leq v_2 \leq \ell\}$ and edge set consisting of all edges joining neighboring vertices except those joining vertices u, v with either $u_1 = v_1 = 0$ or $u_1 = v_1 = \ell + 1$.

Definition 3.3.2. Let $G^*(\ell)$, the *dual* of $G(\ell)$ be the subgrid with vertex set $\{(v_1, v_2) + (\frac{1}{2}, \frac{1}{2}) : 0 \leq v_1 \leq \ell, -1 \leq v_2 \leq \ell\}$ and edge set consisting of all edges joining neighboring vertices except those joining vertices u, v with either $u_2 = v_2 = -\frac{1}{2}$ or $u_2 = v_2 = \ell + \frac{1}{2}$.

Note that every edge $e \in G(\ell)$ has a dual edge $e^* \in G^*(\ell)$ which *crosses* it. We call such e and e^* a *dual pair of edges*. Note also that $G(\ell)$ and $G^*(\ell)$ are isomorphic. Both $G(\ell)$ and $G^*(\ell)$ contain $\ell^2 + (\ell + 1)^2 = 2\ell^2 + 2\ell + 1$ edges.

Definition 3.3.3. The *Paths quorum system* of order ℓ has $n = 2\ell^2 + 2\ell + 1$ elements, and we identify an element in U with a dual pair of edges $e \in G(\ell)$ and $e^* \in G^*(\ell)$. A quorum in the system is a set of elements which contains (elements identified with) the edges of a left-right path in $G(\ell)$ and the edges of a top-bottom path in $G^*(\ell)$.

The *Paths* quorum system of order 3 is depicted in Figure 3.1. The intersection property of the quorum system follows from the following fact:

Fact 3.3.4. *Every left-right path in $G(\ell)$ crosses every top-bottom path in $G^*(\ell)$.*

Naor and Wool proved that the load of the *Paths* quorum system is at most $\frac{2\sqrt{2}}{\sqrt{n}}$ (where $\frac{1}{\sqrt{n}}$ is best possible). Furthermore it is shown that if each node fails with probability smaller than half, then the probability a live quorum exists is at least $1 - e^{-\Omega(\sqrt{n})}$.

3.3.1 Algorithmic Probe Complexity in *Paths*

The analysis of the algorithms we present is based on Theorem 3.3.5 bellow due to Menshikov [93]. The original context of Menshikov's research was Percolation Theory. Consider the infinite

two dimensional grid $\mathbb{Z}_2$ and fix a vertex u. Define $S(k)$ to be the ball of radius k with u at its center, where the distance k is taken according to the grid L_1 metric. The set $\vartheta S(k)$ consists of the vertices in the boundary of the ball. Assume each edge fails with some fixed probability $p > \frac{1}{2}$. Note that since the failure probability is greater than $\frac{1}{2}$, we discuss the case in which *most* edges fail. Define A_k to be the event that there is a path of surviving edges between u and some vertex in $\vartheta S(k)$. The following is Menshikov's Theorem. A good reference for its proof could be found in Grimmett's book [52].

Theorem 3.3.5. *Let $\frac{1}{2} < p \leq 1$ be some failure probability, and let A_k be defined as above. There exists some positive constant $\psi(p)$ such that $\Pr[A_k] < e^{-\psi(p)k}$ for all k.*

Let $\overline{G}(\ell)$ be the dual grid of $G(\ell)$ (just like $G^*(\ell)$), however if an edge in $G(\ell)$ survives then its dual edge in $\overline{G}(\ell)$ fails and if an edge in $G(\ell)$ fails then its dual edge in $\overline{G}(\ell)$ survives. The graph $\overline{G}(\ell)$ is used for the analysis and is not a part of the construction itself. Now, since the failure probability in $G(\ell)$ is smaller than $\frac{1}{2}$, the failure probability in $\overline{G}(\ell)$ is greater than $\frac{1}{2}$, and we can use Theorem 3.3.5. Theorem 3.3.5 bounds the radius of a connected component of $\overline{G}(\ell)$. It states that the radius of a connected component has an exponential decay.

Corollary 3.3.6. *If each edge of $G(\ell)$ fails with probability $p < \frac{1}{2}$, there exists some constant $\delta = \delta(p)$ such that with high probability[1] every connected component of $\overline{G}(\ell)$ is contained in some ball of radius $\delta \log n$ (where the balls are defined by the metric of the grid before failures).*

Proof. Theorem 3.3.5 states that the probability that a ball of radius $\delta \log n$ centered at vertex u does not contain the component of u in $\overline{G}(\ell)$ is less than $e^{-\psi(p)\delta \log n}$. Set δ such that $\delta \cdot \psi(p) \geq 2$. Now for each vertex u this probability is less than $\frac{1}{n^2}$. When applying the union bound over all the n vertices we have that the probability all components are contained in balls of radius $\delta \log n$ is at least $1 - \frac{1}{n}$.

□

A Non Adaptive Algorithm.

We show an algorithm that matches the lower bound of $\Omega(\sqrt{n} \log n)$ for non adaptive probes from Theorem 3.2.1. A left-right path in $G(\ell)$ must *avoid* all the components of surviving edges in $\overline{G}(\ell)$. See Figure 3.2. We describe a non-adaptive algorithm that finds a left-right path, when each element fails with probability $p < \frac{1}{2}$. The case of a top-bottom path is analogous. Choose a horizontal strip of width at least $2\delta \log n + 1$ (where δ is taken from Corollary 3.3.6) and examine all the edges. The algorithm tries to find a left-right crossing within the boundaries of this strip.

Claim 3.3.7. *If each element in the quorum system fails independently with probability $p < \frac{1}{2}$, then after probing non-adaptively $2\ell(2\delta \log n + 1) = \Theta(\sqrt{n} \log n)$ elements, the algorithm finds a quorum with high probability.*

Proof. Corollary 3.3.6 implies that there is no path in $\overline{G}(\ell)$ that crosses the strip top to bottom (otherwise this path is part of a component which can not be contained in a $\delta \log n$ radius ball). By Fact 3.3.4 this implies a left-right path in the strip. See Figure 3.2. □

[1] The term 'with high probability' (w.h.p) means with probability $1 - n^{-\epsilon}$ where ϵ is some positive constant.

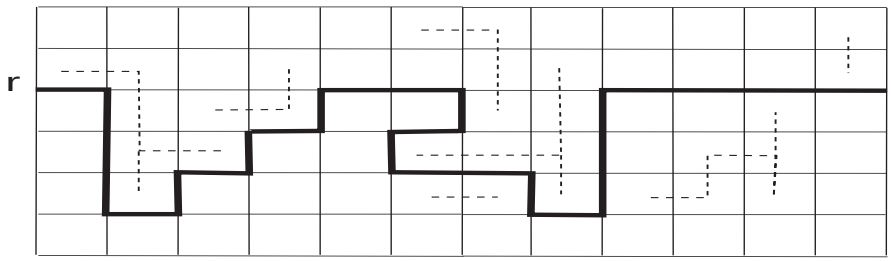


Figure 3.2: A possible quorum in *Paths* after failures. The dashed lines indicate the duals of failed edges. The bold line indicates a left-right path.

Note that while probing $O(\sqrt{n}\log n)$ elements is sufficient to succeed with high probability, Theorem 3.2.1 states that $\Omega(\sqrt{n}\log n)$ probes are necessary to succeed with merely probability $\frac{1}{2}$. As mentioned, since the algorithm is non adaptive it could be implemented in parallel. The actual running time of the algorithm depends on the implementation of the network.

The Load After Failures: Naor and Wool show in [105] (Proposition 5.8) that the load of the *Paths* system is $\Theta(\frac{1}{\sqrt{n}})$ even after failures. In this section we present an efficient non-adaptive algorithm for picking a quorum which meets this bound w.h.p.

Lemma 3.3.8. *If each edge fails with probability $p < \frac{1}{2}$, then there exists a positive constant $\alpha = \alpha(p)$ such that in every strip of width $\alpha \log n$ there exists $\log n$ left-right paths that are edge disjoint.*

Proof. Denote by LR the event that there exists a left-right path in a strip of width $\alpha \log n$ (the constant α will be fixed later). Denote by LR_r the event that there are r *edge disjoint* left-right paths in the strip. Fix some p' such that $p < p' < \frac{1}{2}$. Proposition 5.8 in [105] uses a known result from percolation theory [8] in order to show the following:

$$\Pr_p(LR_r) \geq 1 - \left(\frac{1-p}{p'-p}\right)^r (1 - \Pr_{p'}(LR))$$

When $r = \log n$ we have that $\left(\frac{1-p}{p'-p}\right)^r$ is $O(n^k)$ for some constant k. Since $p' < \frac{1}{2}$, by Corollary 3.3.6 and Claim 3.3.7 we can choose α to be large enough so that $1 - \Pr_{p'}(LR) < n^{-(k+1)}$ and the Lemma follows. □

The strategy of picking a quorum is the following: First pick at *random* a strip and probe all its elements. Find the edge disjoint left-right paths and pick at random one of these paths.

Corollary 3.3.9. *If $p < \frac{1}{2}$ then the load imposed on the elements by the strategy described above is $\Theta(\frac{1}{\sqrt{n}})$.*

Proof. Given a node u, the probability that node u belongs to the randomly chosen strip is $\Theta(\frac{\log n}{\sqrt{n}})$. Lemma 3.3.8 implies that given that u is in the strip, the probability it belongs to the chosen quorum set is at most $\Theta(\frac{1}{\log n})$. As the second event is conditioned upon the first, we may multiply the probabilities and deduce that the load imposed by the strategy is $\Theta(\frac{1}{\sqrt{n}})$ even after failures. □

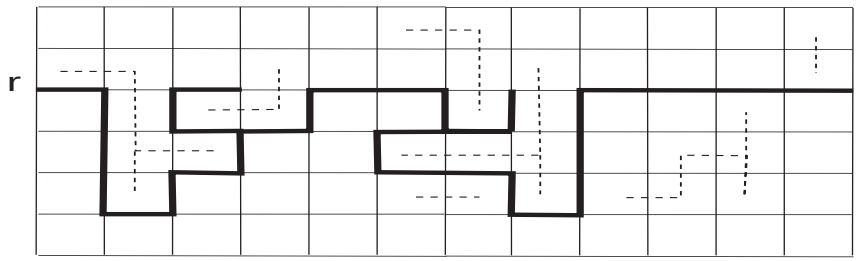


Figure 3.3: The dashed lines indicate the duals of failed edges. The bold line indicates a possible path taken by the adaptive algorithm.

An Adaptive Algorithm for Finding Quorums in *Paths*

Adaptive algorithms can do better than non adaptive ones. Hassin and Peleg presented in [58] a lower bound of $\frac{c}{1-p} + O(1)$ on the expected probe complexity, when c is the cardinality of the smallest quorum set. They proved that for some quorum systems this bound is tight. We note that it is proved in [105] that $\zeta(\mathcal{S}) \geq \max\{\frac{1}{c}, \frac{c}{n}\}$ therefore this lower bound is at best linear in the inverse of the load. In the following we present an adaptive algorithm for the *Paths* quorum system which needs only $\Theta(\sqrt{n})$ probes. It is optimal in the sense that every quorum system with optimal load must have $c = \Omega(\sqrt{n})$. Various adaptive algorithms for quorums were analyzed by Hassin and Peleg [58]. The only quorum system with a better probe complexity is the Crumbling Walls system. This system however suffers from high load. Bazzi presented in [20] the Triangle Lattice quorum system, which resembles *Paths*, and an adaptive algorithm for finding quorums in case of failures. Bazzi proves that its cost of failures (i.e., the communication overhead due to encountering a failed node) is constant. Our algorithm, is an adaptation of Bazzi's algorithm to the grid, and therefore Bazzi's analysis applies in our case and shows the following: let Q be the set of nodes probed by the algorithm until a quorum was found. Let $F \subset Q$ be the subset of which that failed, then there exists a constant α such that $|F| \leq \frac{|Q|}{\alpha}$. This result does not bound the *total number* of probed nodes. Our goal in this section is to show that *with high probability*, the total number of nodes probed is $O(\sqrt{n})$.

We start with a formal description of the algorithm, as before it is sufficient to show how to find a left-right path. The algorithm is a variant of a DFS-search with a specified strategy for picking the next edge to probe. We say that a path *circumvents* a component of $\overline{G}(\ell)$ if it travels along the surface of it; i.e, it travels along edges, the duals of which are adjacent to the component and not part of it. The algorithm would try to find a path which is a straight left-right line. Whenever a component of $\overline{G}(\ell)$ is encountered it would be circumvented. More formally:

1. Choose r at random $1 \leq r \leq \ell$. The search begins at edge r of the left column, and aims to travel along row r.

2. Go to the right. When a failed edge is encountered *circumvent* the component until row r is reached again. If $r \leq \frac{1}{2}\ell$ from above, otherwise from below. Return to row r.

The path in bold presented in Figure 3.3 demonstrates a possible path of the DFS search. The path taken by the algorithm needs to circumvent a component of $\overline{G}(\ell)$ only if it contains the dual of an edge in row r. For each failed edge e of row r define C_e to be the number of edges in the

component of $\overline{G}(\ell)$ that contains the dual of e. If e did not fail then $C_e = 0$. The number C_e is an upper bound on the length of the circumvention the path had to take in order to avoid the failed edge e.

Observation 3.3.10. *The length of the path taken by the algorithm is at most $\ell + \sum C_e$ where the sum is taken over the edges of row r.*

Theorem 3.3.11. *The probe complexity of the algorithm is $\Theta(\ell) = \Theta(\sqrt{n})$ with high probability (where the probability is taken over the occurrence of faults).*

Proof. Assume that the random starting point selected in Step (1) of the algorithm is a starting point of some left-right path of the grid. By Lemma 3.3.8 we know that the probability of this is constant. Thus we repeat the procedure above, until a good starting point is found. We need to show that all the circumventions taken in Step (2), i.e., $\sum C_e$, accumulate to no more than $\Theta(\ell)$. Fix some edge e on row r, and let vertex u belong to its dual edge. Let C_u be the number of edges in the component of u in $\overline{G}(\ell)$. Let A_u denote the diameter of that component. Since the vertex u is adjacent to the dual of e it holds that $C_u \geq C_e$. The grid topology implies that if $C_u \geq k$ then $A_u \geq \frac{1}{2}\sqrt{k}$. We have (by Theorem 3.3.5) that for some $\psi(p) > 0$

$$\Pr[C_u \geq k] \leq \Pr[A_u \geq \tfrac{1}{2}\sqrt{k}] \leq e^{-\psi(p)\sqrt{k}} \tag{3.4}$$

$$E[C_u] = \mu \leq \sum_{k=1}^{\infty} k \cdot e^{-\psi(p)\sqrt{k}} = O(1) \tag{3.5}$$

The algorithm may need to avoid at most ℓ components of $\overline{G}(\ell)$. By linearity of expectation the expected probe complexity of the algorithm is $\Theta(\ell)$. To show that this sum is $\Theta(\ell)$ with high probability we need a slightly different argument. Divide the grid into $(\frac{\ell}{\delta \log n})$ vertical strips each of width $\delta \log n$, where δ is taken from Corollary 3.3.6. Each strip is wide enough such that w.h.p it is wider than any component of $\overline{G}(\ell)$. Assume this high probability event occurs. Define $X_i \stackrel{def}{=} \sum C_e$ where the sum is taken over edges of row r and strip i. The length of the path the algorithm took is at most $\ell + \sum X_i$.

Lemma 3.3.12. *$E[X_i] \leq \mu\delta \log n$ and w.h.p for all i we have $X_i \leq 2\delta^2 \log^2 n$.*

Proof. The width of the strip is $\delta \log n$ and the expected size of each component is μ, therefore by linearity of expectation $E[X_i] \leq \mu\delta \log n$. By Corollary 3.3.6 we know that w.h.p all the components are contained in a $\delta \log n$ radius ball. Therefore w.h.p all the components are confined into a rectangle of area $2\delta^2 \log^2 n$. which proves the second claim. □

Define $I_\sigma = \{1 \leq i \leq \frac{\ell}{\delta \log n} : i \mod 3 = \sigma\}, \sigma \in \{0, 1, 2\}$.

Lemma 3.3.13. *Conditioned on the event that all the components are of diameter $O(\log n)$, which by Corollary 3.3.6 occurs with high probability, the set $\{X_i\}_{i \in I_\sigma}$ consists of independent random variables.*

Proof. If all components are of small diameter, then every connected component of $\overline{G}(\ell)$ belongs to at most two strips. Therefore X_i depends only upon the probes of edges in strips $i-1, i, i+1$. This means that X_i, X_{i+3} are independent. □

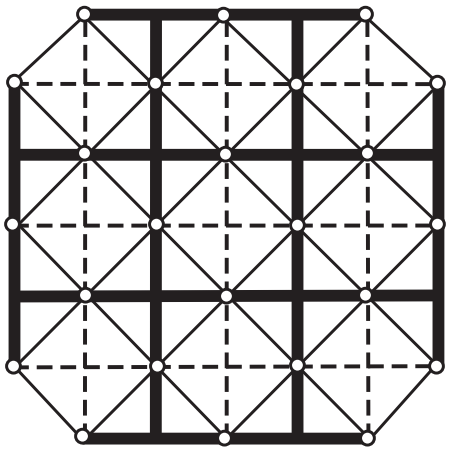


Figure 3.4: The line graph of a 4×4 grid and its dual.

By using the appropriate Chernoff Hoefding bound (cf. [51] page 17) we have

$$\Pr\left[\sum_{I_\sigma} X_i - \sum_{I_\sigma} E[X_i] \geq t|I_\sigma|\right] \leq 2\exp\left(-\frac{2t^2|I_\sigma|}{(2\delta^2\log^2 n)^2}\right)$$

Since $|I_\sigma|$ is in the order of $\frac{\sqrt{n}}{\log n}$, the probability that there is a large deviation decays exponentially fast. In particular setting t to be some large enough constant implies that $\Pr[\sum_{I_\sigma} X_i > \Theta(\ell)] \leq \frac{1}{n^2}$ for $\sigma \in \{0,1,2\}$. Now we apply the union bound over the high probability events of Corollary 3.3.6 and Lemmas 3.3.12 and 3.3.13, which means that with high probability the probe complexity of the algorithm is $\Theta(\ell) = \Theta(\sqrt{n})$. This concludes the proof of Theorem 3.3.11. □

Network implementation: In order to calculate the *actual* running time and message complexity of these algorithms we need to take into consideration the topology and implementation of the network over which the quorum system is defined. The most natural network topology to consider is that of $G(\ell), G^*(\ell)$ themselves. Each node is associated with a pair of dual edges, and is connected to the nodes that are associated with edges that are adjacent to its own edges. In other words, the topology of the network is the *line graph* of the two dimensional grid. In Figure 3.4 the thick solid edges belong to the line graph of $G(\ell)$, the dotted edges belong to the line graph of $G^*(\ell)$ and the diagonal edges belong to both. A quorum set therefore is composed of nodes that form a left-right path using the solid horizontal, vertical and diagonal edges and a left-right path using the dotted and diagonal lines. In this implementation the message complexity and the time complexity of the adaptive algorithm are indeed $\Theta(\ell)$. The non-adaptive algorithm can probe its chosen strip in parallel, and achieve a running time of $\Theta(\ell)$ and a message complexity of $\Theta(\sqrt{n}\log n)$. Other data structures that are implemented on the network might support the implementation of the quorum system. For instance if the network implements a DHT then the DHT could be used for probing the strip in parallel and the time complexity would reduce to $\Theta(\log n)$ with an extra logarithmic factor in the message complexity.

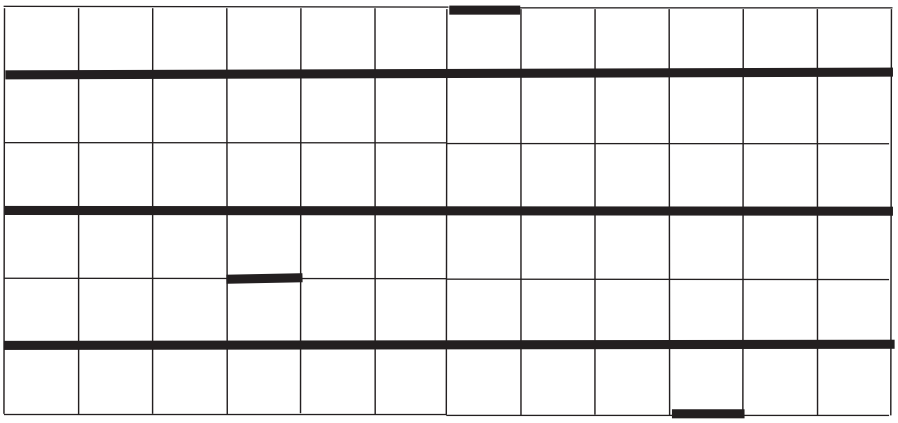


Figure 3.5: A possible sample from the worst case distribution of failed edges. The bolded edges have failed. The third row is an open path.

Worst case model: Assume an adversary is given the possibility to crash a constant fraction of the elements. It is easy to see that an adversary can 'kill' all the short paths, and leave only paths of length $\Omega(\ell^2) = \Omega(n)$. However an adversary may force any algorithm (even probabilistic) to probe $\Omega(n)$ elements, even if we are guaranteed that there exists a short left-right path. We sketch the proof using Yao's minimax principle (cf. [97]). We need to supply a distribution of the *inputs* such that every deterministic algorithm would need to probe an expected $\Omega(n)$ elements. The distribution over inputs is as follows:

1. Kill every line of even index.
2. From the remaining lines choose at random one which would remain alive.
3. Kill each remaining line by choosing at random one element from it and deleting it.

An example of a possible input is seen in Figure 3.5, where the third row from the top is the only surviving row. Now every *deterministic* algorithm needs to find the line that survived. Every such algorithm will need to probe $\Omega(\ell)$ lines, each of these lines should be probed $\Omega(\ell)$ times. All in all every deterministic algorithm would probe on expectation $\Omega(\ell^2)$ edges. We conclude that for every algorithm (deterministic or randomized) there is an input, for which the expected probe complexity of the algorithm is $\Omega(n)$. Peleg and Wool analyze in [108] the probe complexity of several quorums under the model of adversarial deletion. They show several lower bounds, all of which turn to be $\Omega(\ell)$ in the *Paths* system. Note that even though the algorithmic probe complexity is high, the *cost of failures* (as were defining by Bazzi [20]) is a constant.

3.4 The *Dynamic Paths* Quorum System

In this section we suggest a quorum system that operates in a dynamic model, where nodes may join and leave. The applications of quorum systems in a dynamic setting were considered in a wide range of papers cf. [1],[59],[65]. Previous constructions of dynamic quorums focused on

implementing quorum systems in a dynamic environment and designing algorithms that allowed a group of nodes to *form* a new quorum in a consistent way (cf. [61],[79],[111], [54],[81]). We focus on the *combinatorial* properties of dynamic quorums. Our goal is to design dynamic quorums that enjoy low load, high availability, low probe complexity and that scale gracefully in respect to these parameters. Stojmenović and Peña [121] suggest a location based dynamic quorum system for use in ad-hoc wireless networks. The system is composed of North-South and West-East paths which are constructed dynamically according to the physical location of the nodes. Our work differs by assigning *virtual* coordinates to nodes, thus using the quorum system in a more general setting. We then provide a rigorous analysis in which the combinatorial properties (load, availability, integrity) of the system are analyzed.

3.4.1 The Quorum System

The good properties of the PATHS quorum system motivates us to design a *dynamic version* of the PATHS system. The main idea is to substitute the grid with the *continuous* unit square $[0,1) \times [0,1) \subset \mathbb{R}^2$. Now we use the continuous-discrete approach in the two-dimensional case as described in Section 2.7. In the *Dynamic Paths* quorum system, a quorum set is the union of (elements identified with) the vertices (generators) that form a left-right path and a top-bottom path in the Delaunay graph.

Load: We upper bound the load by analyzing a specific distribution over quorum sets: choose at random two points (x, y) in the interval $[0, 1)$. Now pick the quorum set that is composed from all the cells that intersect the horizontal line x and the vertical line y. An example of a quorum set is depicted in Figure 3.8. The bound on the projection of a cell in Theorem 2.7.3 implies that with high probability the load imposed by this strategy is at most $\Theta(\frac{\sqrt{\log n}}{\sqrt{n}})$.

Availability: Planar duality implies that if the locations of the generators were picked uniformly at random, and the failure probability is $\frac{1}{2}$ then with probability $\frac{1}{2}$ a left-right path exists. A rough outline of the argument is as follows: If there is no left-right crossing, then there must be a top-bottom crossing of *failed* Voronoi cells. The procedure of creating the Voronoi diagram is symmetric and imposes the same probability over a top-bottom and a left-right crossing. Therefore when $p = \frac{1}{2}$ the probability that a crossing exists is equal to the probability a crossing does not exist. This suggests strongly that the critical probability (i.e. the value of p in which the probability there is a crossing when $n \to \infty$ jumps from 0 to 1) is at $\frac{1}{2}$. Indeed recently Bollobas *et al* [24] prove that this is the case. Another important result in [24] is that when $p > \frac{1}{2}$, i.e. most of the cells fail, the size of the connected components has an exponential decay. In other words Bollobas *et al* prove a theorem analog to Menshikov's Theorem 3.3.5. This implies that the same analysis done when specifying the availability of *Paths* should hold for *Dynamic Paths* as well.

Integrity: It is necessary that nodes save some information about the quorum sets that were used. A quorum set is associated with a path. Every time a quorum is used, a node that participates in the quorum should remember the identity of the nodes before and after it in the path. When a new node joins the system either the quorum set grows or the load should be divided evenly between the new quorums. Figure 3.6 demonstrates the process. Figure (a) shows the Voronoi diagram before the entrance of v. Figure (c) demonstrates the case where v is added to quorum

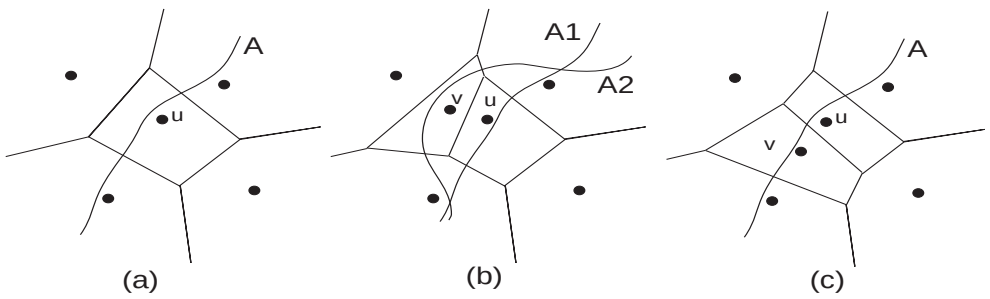


Figure 3.6: Maintaining the integrity of the quorum system.
When adding a node either the load of quorum A is split between quorums A_1, A_2 as seen in (b), or A grows, as seen in (c).

A. Figure (b) shows the case where the responsibilities of quorum A (represented by the line in bold) should now be split between quorums A_1, A_2. If for instance the application of the quorum system is mutual exclusion, and quorum A is currently active, then nodes u, v should decide among themselves which one of them remains active, and inform their neighbors. If the quorum system is used for replication of data, then the procedure is slightly more delicate. Each data item is associated with a quorum set. Nodes u, v should divide among themselves the data items that were previously associated with quorum A, and of course inform their neighbors.

Algorithmic probe complexity: The probing algorithms that were described in Section 3.3 have obvious analogs in the *Dynamic Paths* system. In order to prove that the probe complexity of the non-adaptive and adaptive algorithms is $\Theta(\sqrt{n}\log n)$ and $\Theta(\sqrt{n})$ respectively, we need an analog for Theorem 3.3.5; i.e., we need that for a small failure probability, the radius of a component of *failed* cells would decay in sub-exponential rate. Until recently such a theorem was unknown. The recent paper by Bollobas *et al* [24] mentioned above, proves exactly this. So the performance of the algorithms could be analyzed in the same manner as in Section 3.3 and the *Dynamic Paths* quorum system enjoys excellent probe complexity.

3.4.2 A Smooth Voronoi Diagram

The Single Choice Join algorithm is simple and offers optimal availability and probe complexity. It has the disadvantage though that some of the Voronoi cells are quite big, causing an increase in the load and the degree. The load of the system is proportional to the size of the projection of the cells over the axis lines. Theorem 2.7.3 bounds the size of the projection (and therefore the load) by $O(\sqrt{\frac{\log n}{n}})$, in the case where the location of the nodes is chosen uniformly and randomly in $[0,1)\times[0,1)$. Furthermore the existence of large cells enlarges the maximum degree in the network to be logarithmic.

As seen in Section 2.7, a more sophisticated and coordinated procedure for choosing the location upon entrance reduces the size of the largest cell and creates a *smooth* Voronoi diagram. In a *smooth Voronoi Diagram* every cell is contained in a square of area $\Theta(\frac{1}{n})$ and therefore its projection on the axis lines is $O(\frac{1}{\sqrt{n}})$. This implies that the load of the quorum system would be optimal and the maximum degree is constant. A slight difficulty arises when trying to prove the high availability

and low probe complexity of the algorithms. We know that both the grid and a random Voronoi diagram have high availability and low probe complexity whenever the failure probability is smaller than $\frac{1}{2}$. It is very likely therefore that a smooth Voronoi diagram has the same properties. We can prove a somewhat weaker claim, namely that the availability and probe complexity are as good as in *Paths*, as long as the failure probability is at most some small constant (smaller than $\frac{1}{2}$).

Intuitively if the Voronoi diagram is smooth then 'it looks like a grid' and therefore theorems that are correct for the grid should apply for the diagram. The technique we use follows this intuition, though it is rather delicate. We use domination by product measures as shown by Liggett *et al* in [77]. We need some definitions from probability theory. In the following we define the necessary definitions and sketch the idea of the proof. A good exposition of the notions we use appears in Grimmett's book[52]. The discussion below follows it.

Domination by Product Measures

We begin by defining stochastic domination in our context. Say we have a finite set S and a state space $\Omega = \{0,1\}^S$. The set S may be the set of edges in a two dimensional grid and Ω the set of configurations when some of the edges fail. Given $\omega_1, \omega_2 \in \Omega$ we say that $\omega_1 \leq \omega_2$ if $\forall s \in S$ $\omega_1(s) \leq \omega_2(s)$. In our case $\omega_1 \leq \omega_2$ if all the surviving edges in ω_1 have also survived in ω_2.

Given a function $f : \Omega \to \mathbb{R}$ we say that f is *increasing* if

$$\omega_1 \leq \omega_2 \quad \Rightarrow f(\omega_1) \leq f(\omega_2).$$

For instance the function that assigns the value 1 to a configuration that contains a left-right crossing and 0 otherwise, is an *increasing function*.

Now, given two probability measures on Ω, μ and ν we shall say that μ *stochastically dominates* ν - and write $\mu \succeq \nu$ - if for any increasing function f we have $E_\mu(f) \geq E_\nu(f)$. This is a very strong condition which amounts to saying that in every possible way, μ puts more mass on bigger elements of Ω than ν does. In case that f is defined as above, it means that the probability there exists a left-right path is larger in μ than it is in ν. A canonical example for domination is the following: Assume we have a two dimensional grid. Denote by π_p the product measure with probability p, i.e., the case in which each edge fails independently with probability $1-p$. It is intuitive (though requires proof) that $\pi_{p_1} \succeq \pi_{p_2}$, when $p_1 \geq p_2$.

The analysis of *Paths* used various bounds on increasing events on the *product measure* over the grid. Our approach would be to show that the process of randomly failing cells in a smooth Voronoi diagram *dominates* a product measure on the grid, thus lower bounding the probability there exists a left-right path in the Voronoi diagram.

Let T be a smooth Voronoi diagram with n generators and assume that each cell survives with probability $p > \frac{1}{2}$ and fails with probability $1-p$, independently from all other cells. Now construct a $\sqrt{n} \times \sqrt{n}$ grid called G *on top* of the Voronoi diagram, as shown in Figure 3.7. We say that an edge $e \in G$ failed iff it intersects a failed cell of T. Let X_e be the indicator of the state of e (i.e., $X_e = 1$ iff e survived). Now $\Pr[X_e = 1]$ is exactly p to the power of the number of cells it intersects. However since T is smooth, we know that this power is bounded by some constant, therefore there exists some $p' < p$ independent of n, such that for all $e \in G$, $\Pr[X_e = 1] \geq p'$. Assume that p was large enough such that $p' > \frac{1}{2}$.

Observation 3.4.1. *If there exists a left-right crossing of survived edges in G then there exists a left-right crossing of survived Voronoi cells in T (i.e., a crossing in the Delaunay graph).*

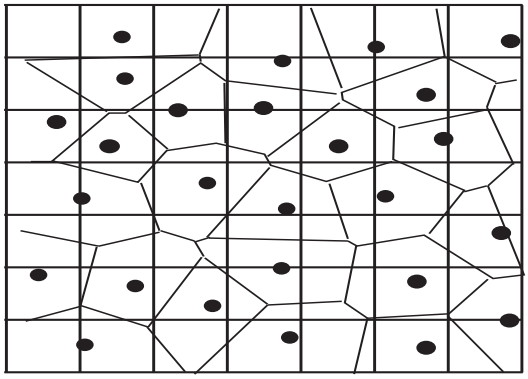


Figure 3.7: The grid G is put on top of the diagram T.

Since $p' \geq \frac{1}{2}$ one is tempted to use known results from percolation theory that show that the probability of a crossing is very high, as was used in [105] to prove the availability of *Paths* and as was used perviously to prove the low probe complexity of *Paths*. The problem is that the random variables $\{X_e\}_{e \in G}$ are *not mutually independent*. In particular, if two edges are contained in the same cell in T, then the state of both of them is determined by the state of that cell. The key observation is that since T is smooth, X_e is independent from all but *a constant number* of other edges. Let μ be the probability measure thus defined on $\{X_e\}_{e \in G}$. Liggett *et al* show in [77] that in this case μ *dominates* the product measure over the edges of G for some other value $r' \leq p'$. Theorem 1.3 in [77] could be stated in our case as follows:

Theorem 3.4.2. *Let μ be some probability measure over the set of configurations of the edges of G. Assume that each edge in G survives with probability at least p', and that the state of each edge is dependent on the state of at most k other edges for some constant k. Then there exists some r' which is a function of p', k and independent of n such that $\mu \succeq \pi_{r'}$. Furthermore by increasing p', r' could be made arbitrarily close to* 1.

Intuitively speaking Theorem 3.4.2 states that if we have a two dimensional grid, and each edge fails 'almost' independently from all other edges, then by reducing the failure probability, we may think as if each edge failed independently. Note that the existence of a left-right path in the grid is an increasing event. The diameter of a connected component in the dual graph (which is bounded in Theorem 3.3.5) is also an increasing function. Theorem 3.4.2 implies that by reducing the failure probability, we may use these theorems to bound those random variables in the smooth Voronoi diagram.

Denote by $G_\mu(p')$ the random graph induced by $\{X_e\}_{e \in G}$. Denote by $G_\pi(r')$ the random graph induced by the product measure with probability r'.

Corollary 3.4.3. *Let p' be close enough to* 1. *There exists some $r' \leq p'$ independent from n, such that the probability there exists a crossing in $G_\mu(p')$ is at least the probability there exists a crossing in $G_\pi(r')$, and the probability a component of the dual $G_\mu(p')$ is of diameter k, is at most the probability a component of the dual of $G_\pi(r')$ is diameter k. Furthermore by increasing p', r' could be made arbitrarily close to* 1.

Corollary 3.4.3 is directly used to analyze the availability and probe complexity of *Dynamic Paths*:

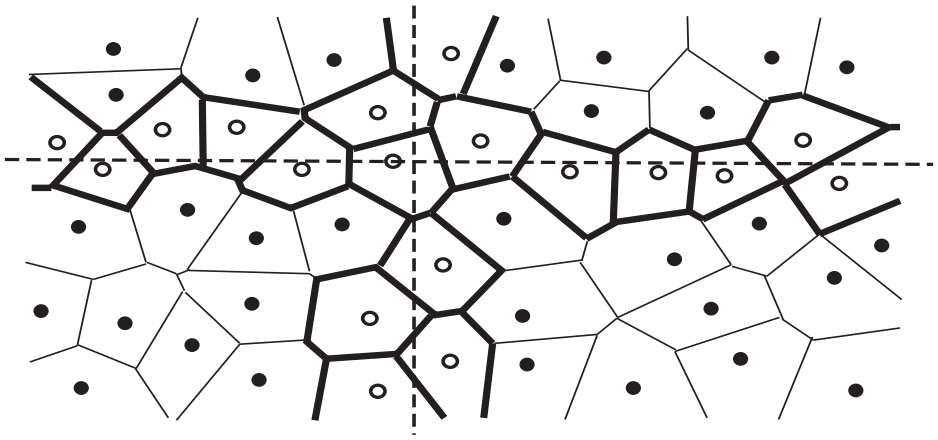


Figure 3.8: An example of a quorum in *Dynamic Paths*.
The cells that belong to the quorum set are the ones that intersect the dashed lines.

Theorem 3.4.4. *Let T be a smooth Voronoi diagram, and let $\mathcal{S}$ be the* Dynamic Paths *quorum system derived by it. Then the load of the system $\zeta(\mathcal{S})$ is $O(\frac{1}{\sqrt{n}})$. There exists some $\frac{1}{2} < p_c < 1$ such that for $p_c < p < 1$, if each node fails independently with probability $1-p$ then the following hold:*

1. *The probability a live quorum set exists is $1 - e^{-\Omega(\sqrt{n})}$*
2. *The non-adaptive algorithmic probe complexity is $O(\sqrt{n}\log n)$ w.h.p.*
3. *The adaptive algorithmic probe complexity is $O(\sqrt{n})$ w.h.p.*

3.4.3 A simpler Quorum System

A possible simplification of the Dynamic Paths system is the following: Define a quorum set to be all the (elements identified with) cells that intersect the same horizontal and vertical line (see Figure 3.8). This quorum system is a dynamic adaptation of a quorum system suggested by Maekawa [82]. A slight improvement was suggested by Agrawal *et al* in [6] where instead of looking at horizontal and vertical lines, they examine diagonal lines that resemble the paths of billiard balls. Theorem 2.7.3 implies that the load of these quorum systems is $\Theta(\sqrt{\frac{\log n}{n}})$. The integrity of these systems could be maintained by associating each quorum set with the numeric value of the vertical and horizontal lines, thus the implementation is simpler. The main drawback of these systems is their low availability. If each node fails with probability $\Theta(\frac{\log n}{\sqrt{n}})$, then with high probability no quorum set survives.

3.5 Conclusion and Open Questions

The main open problem is to improve the load of the *Dynamic Paths* quorum system so that it matches the load of *Paths*. The load of *Dynamic Paths* is determined by the size of the projection of cells over the axis lines. The simple Join algorithm in which the location of each generator is

random guarantees that the projection of all cells is at most $O(\frac{\sqrt{\log n}}{\sqrt{n}})$. It is interesting to find other (perhaps more sophisticated) Join algorithms that guarantee a better load. The cyclic presented in Section 2.6 applies for the one dimensional case only. It would be interesting to find a two dimensional analog to that algorithm. Some work in this direction was done in [5], however they considered splitting the plain into rectangles (as in CAN) and not a Voronoi diagram.

A better understanding of percolation theory over Voronoi diagrams would improve the analysis of the algorithms. In particular it is important to bound the probability of a diameter k component in a percolation with $p < \frac{1}{2}$. A 'Menshikov style' theorem of this sort that states that this probability is exponentially small in k, would imply a $\Theta(\log n\sqrt{n})$ algorithmic probe complexity for *Dyanmic Paths* even for the simple random Join algorithm.

Conclusion: The *Paths* quorum system is shown to have excellent adaptive and non-adaptive probing algorithms. It was previously known that the *Paths* system has optimal load and availability, thus the *Paths* system offers excellent balance between different quality measures. This makes *Paths* a natural candidate for an adaptation into a dynamic setting. Applying the continuous-discrete technique results with the *Dynamic Paths* quorum system which is scalable and operates in a dynamic setting. Dynamic quorum systems were used by Nadav and Naor [98] for designing fault tolerant storage systems. *Dynamic Paths* maintains the good qualities of the *Paths* system. Its low load, high availability and simple probing algorithms makes it an excellent candidate for an implementation of dynamic quorums.

Chapter 4

The Neighbor of Neighbor Algorithm

Summary: Several peer-to-peer networks are based upon randomized graph topologies that permit efficient GREEDY routing, e.g., randomized hypercubes, randomized Chord, skip-graphs and constructions based upon small-world networks. In each of these networks, a node has out-degree $O(\log n)$, where n denotes the total number of nodes, and GREEDY routing is known to take $O(\log n)$ hops on average. Our contribution is twofold. First we investigate the limitations of GREEDY routing and establish lower-bounds for GREEDY routing for these networks, then we present and analyze the *Neighbor-of-Neighbor* (NoN)-GREEDY routing. The idea behind NoN, as the name suggests, is to take a neighbor's neighbors into account for making better routing decisions.

The following picture emerges: Deterministic routing networks such as hypercubes and Chord have diameter $\Theta(\log n)$. This means that GREEDY routing is optimal in the sense that its routing distance is at most (approximately) the diameter, yet networks with average degree of $O(\log n)$ may have diameter $O(\frac{\log n}{\log \log n})$. Randomized routing networks such as skip-graphs, randomized hypercubes, randomized Chord, and constructions based upon small-world percolation networks, have diameter $\Theta(\log n / \log \log n)$ with high probability. In all of these networks, GREEDY routing fails to find short routes, requiring $\Omega(\log n)$ hops with high probability. Surprisingly, the NoN-GREEDY routing algorithm is able to diminish route-lengths to $\Theta(\log n / \log \log n)$ hops, which is asymptotically optimal.

4.1 Small World Networks

Randomized network constructions that model the *Small-World Phenomenon* have recently received considerable attention. A widely-held belief pertaining to social networks is that any two people in the world are connected via a chain of six acquaintances (*six-degrees of separation*)[1]. The behavior of such networks has been investigated extensively by researchers from diverse set of disciplines including: the social sciences, physics, computer science and webologists. These investigations consist of either checking the existence of the phenomenon in various setting or coming up with models to explain it. The quantitative study of the phenomenon started with Milgram's [94] experiments in 1960's, asking people to send letters to unfamiliar targets only through acquaintances. Milgram's experiments and the work by Pool and Kochen [110] confirmed that often random pairs of individuals are indeed connected by short chains. Watts and Strogatz [123] claimed that the Small-World Phenomenon is common in a variety of different realms and suggested modeling the phenomenon by analyzing some distribution over random graphs.

[1] According to Barabási [18] this idea may have its origins in a short story "Chains" by the Hungarian writer Frigyes Karinthy from 1929; this idea has been retold and recast many times since then, in the literature, popular press as well as scientific studies.

The study of the algorithmic or routing perspective of this phenomenon was initiated by Kleinberg [70],[69], who pointed out that the small world experiments showed not only that short paths exist, but that *people can find such paths based on local information.* To model the routing aspects of the Small-World Phenomenon, Kleinberg considered 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"). In particular, Kleinberg considered a two dimensional $n \times n$ 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, i.e, the probability of establishing an edge (x, y) is proportional to $||x - y||^{-2}$, where $||x - y||$ stands for the grid's L_1 distance. With GREEDY routing, the path-length between any pair of nodes is $O(\log^2 n)$ hops, w.h.p. Local knowledge available to a node suffices for GREEDY routing – a message is forwarded along that out-going link which takes it *closest* to the destination. Barrière *et al* [19] showed that GREEDY routing requires $\Omega(\log^2 n)$ hops for Kleinberg's construction. The diameter of small world graphs is shorter and is $\Theta(\log n)$ on expectation [90]. Thus, GREEDY routing is sub-optimal and it is desirable to find routing schemes that route along shorter paths.

Kleinberg's results can have various interpretations: it could be thought of as an explanation of how people in the chain letter experiments behaved (this is a descriptive approach). Alternatively, it could be seen as suggesting a routing strategy. While sending letters to unknown targets is not the most useful activity, the problem is related to routing in peer-to-peer networks, i.e. networks where nodes join and leave the system dynamically. In various P2P systems nodes are assigned labels that are interpreted as points on some $d-$dimensional space; links are added to close neighbors and some links are added to far away ones. Hence the hope is that lessons learned for the small world graphs may be applicable in the peer-to-peer environment. Indeed the works of Aspnes *et al* [14] and Manku *et al* [87] apply intuitions from small worlds into peer-to-peer constructions. We shall show further adaptations in this work.

Peer-to-Peer Networks

The topology of P2P networks can be classified into two categories – deterministic and randomized. In deterministic P2P networks the topology is a function of the id's of the nodes. Typically they are based upon classical parallel inter-connection networks, such as the hypercube and its variants, butterflies or De-Bruijn graphs. Examples include the Distance Halving DHT and many more e.g [115, 128, 120, 2]. In randomized networks, as the name suggests, randomization is used to determine the topology of the network. Natural examples include skip-graphs [16, 57], the randomization of hypercubes [53, 28] and the randomization of Chord [127, 53]. All of which have a node degree of $O(\log n)$. Other examples are networks based on Kleinberg's construction such as Symphony [87] [14]. In these networks the out-degree of each node is bounded by a constant. Among the various P2P routing networks, skip-graphs are unique in that node identifiers (or "keys" associated with nodes) can be drawn from an arbitrary ordered domain, e.g., the set of character strings. This property makes skip-graphs the only P2P routing network that naturally supports *prefix-search.* Other P2P routing networks assume that nodes are assigned identifiers that are drawn uniformly from the unit interval $[0, 1)$.

Many P2P networks 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 $\frac{1}{||i-j||^d}$, independently of all other edges. We call this network a *small-world percolation network.* The small-world perco-

lation network has its antecedents in classical "long range percolation" models. We outline a brief history at the beginning of Section 4.2.

The routing scheme suggested for all the above networks is GREEDY; i.e., each node routes the message to its neighbor which is closest to the target. The GREEDY algorithm is appealing to use since it is based on local information only and is very simple (both conceptually and implementation-wise). The main disadvantage of GREEDY is that often it routes along paths that are much longer than the shortest paths in the network. The Neighbor-of-Neighbor (NoN) greedy algorithm is meant to overcome this problem. The idea underlying NoN is to allow a node to gain knowledge of its neighbor's neighbors for assistance in making better routing decisions. . Our work addresses two questions:

(a) When does GREEDY routing route along (approximately) shortest paths?

(b) What is the role of look-ahead (or Neighbor of Neighbor) upon GREEDY routing?

4.1.1 Our Contributions

In a network with k out-going links per node, the average length of shortest paths is $\Omega(\log n/\log k)$. Therefore, with $O(\log n)$ links per node, it *might* be possible to route in $O(\log n/\log\log n)$ hops, and it *might* be possible to route in $O(\log n)$ hops in Kleinberg's construction. The known upper bounds for GREEDY routing are sub optimal in this sense. The main contribution of this work is to show that in many cases GREEDY routing is indeed asymptotically sub-optimal, while the NoN-Greedy algorithm which uses just one level of look-ahead is asymptotically optimal. In particular we show the following:

Upper bounds: We show that NoN-GREEDY routing, which fixes two hops of a route (by taking the neighbors of neighbors of a node into account), is optimal for the small-world percolation networks and requires $\Theta(\log n/\log\log n)$ hops, w.h.p. (Section 4.2). The same upper bound is established for randomized-hypercubes and randomized-Chord (Section 4.3) and for skip graphs (Section 4.4). Thus skip-graphs are the only degree-optimal P2P network that supports *prefix search*. In Section 4.3 we also analyze Kleingberg's construction (Symphony) and show that the NoN algorithm is asymptotically better than GREEDY yet not optimal.

The asymptotical analysis is accompanied by simulations which show that for network sizes ranging from 2^{12} to 2^{24} nodes, NoN-GREEDY routes are 40% to 48% shorter than GREEDY routes in all of these topologies (Section 4.6).

Lower bounds In Section 4.5 we show that GREEDY routing requires $\Omega(\log n)$ hops on average in small world percolation graphs and in each of the following randomized P2P networks: skip-graphs, randomized-Chord, randomized-hypercube, and Symphony with $k = \Theta(\log n)$ per node.

4.1.2 Related Work

The tradeoff between the average path length and the out-degree of nodes is of fundamental interest to designers of P2P routing networks. Hypercubes and Chord offer average paths of length $\Theta(\log n)$ with $\Theta(\log n)$ links per node with GREEDY routing (optimal routes in Chord were identified by Ganesan and Manku [45]). Skip graphs, Randomized-hypercubes and randomized-Chord

were known to offer routes of length $O(\log n)$ with GREEDY routing. Among the randomized P2P networks, Viceroy [84] offers routes of length $\Theta(\log n)$ w.h.p. with only $O(1)$ links per node. A randomized construction in [86] combines ideas from Viceroy with Kleinberg's construction to arrive at a network that routes in $\Theta(\log n/\log k)$ hops w.h.p., with k links per node.

Networks based on De-Bruijn graphs [99, 62, 41] offer an optimal tradeoff between degree and path length, in particular for $O(\log n)$ links per node the routes are of length $O(\log n/\log\log n)$. The De-Bruijn networks are significantly simpler than Viceroy and the construction in [86], yet the routing protocol in De-Bruijn graphs is not GREEDY – it is based on numeric computations on labels of nodes. Recently Abraham *et al* [4] presented a graph based on the Butterfly network in which when the degree is d, GREEDY routes along paths of length $O(\log n/\log d)$.

Overall, two classes of networks are known to have optimal route lengths with respect to the degree, for instance route in $\Theta(\log n/\log\log n)$ hops with $\Theta(\log n)$ links per node: De-Bruijn networks and butterfly networks. The P2P implementation of these networks requires that keys are *random*, thus unlike skip-graphs there is no natural way for keys to carry *semantic* meaning. The results of this paper add a third class – "randomized small-world networks". We hope that our results inspire further investigations into the general properties of these networks.

The basic idea of the NoN-GREEDY approach is drawn from two sources. A paper by Coppersmith *et al* [33] uses the neighbors-of-neighbors approach, though not in an algorithmic perspective. They use the idea to establish that the diameter of small-world percolation networks on n nodes is $O(\frac{\log n}{\log\log n})$ w.h.p. NoN-GREEDY *routing* was first used (under the name "GREEDY with 1-LOOKAHEAD") by Manku *et al* [87] as a heuristic for Symphony, a randomized P2P network. Fraigniaud *et al* [42] recently analyzed other variants of GREEDY algorithms in Kleinberg's model, when each node is aware of the long-range contacts of the $\log n$ nodes which are closest to it. They show that a variant of GREEDY which routes in expected $\Theta(\log^{1+\frac{1}{d}} n)$ hops (when d is the dimension of the mesh). Aspnes *et al* [14] established lower bounds for GREEDY over a general family of randomized networks under the assumption that each "long-range" link is drawn from the same probability distribution. Lebhar and Schabanel [74] present a routing algorithm which is not greedy and improves over the simple greedy algorithm.

4.1.3 The NoN-GREEDY Routing Algorithm

We introduce the main object of our investigation, the NoN-GREEDY Routing Algorithm, in Figure 4.1. We assume the existence of a metric on the labels of nodes.

Algorithm for routing a message to node t.

1. Assume the message is currently at node $u \neq t$. Let $w_1, w_2, \ldots, w_k$ be the neighbors of u.
2. For each w_i, $1 \leq i \leq k$, find z_i - the closest neighbor to t. Let j be such that z_j is the closest to t among $z_1, z_2, \ldots, z_k$.
3. Route the message from u via w_j to z_j.

Figure 4.1: The NoN-GREEDY Algorithm. Some metric over the labels of nodes is assumed.

In the NoN-GREEDY algorithm, w_j may not be the neighbor of u which is closest to t. The algorithm could be viewed as a greedy algorithm on the *square* of the graph – a message gets routed to the best possible node among those at distance two.

4.2 NoN in Small-World Percolation Graphs

Definition 4.2.1. A "small-world percolation network" of dimension d is a finite graph whose vertex set is associated with the $d-$dimensional mesh. The probability that (u, v) is an edge is $\frac{1}{||u-v||^d}$ and is independent from all other edges, and $||u - v||$ stands for the mesh L_1 distance between u and v.

Small-world percolation networks originate from a classical percolation model called "long range percolation". In that model, nodes lie on an infinite grid and an edge is put between a pair of nodes with some positive probability. The question of existence of infinite components was considered by Schulman [118], Aizenman and Newman [9] and Newman and Schulman [106], where the one dimensional grid $\mathcal{Z}$ is studied and edges (i, j) are selected with probability $\beta/\|i - j\|^s$ for some values β, s.

Benjamini and Berger [22] 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 values β, s. Coppersmith *et al* [33] extended the model to multiple dimensions: a $d-$dimensional mesh where an edge (u, v) is selected independently with probability $1/\|u - v\|^d$. Coppersmith *et al* [33] 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. We now show that Non-GREEDY routing results in paths of length $\Theta(\log n/ \log\log n)$ w.h.p.

Theorem 4.2.2. *Given two nodes s, t in a $d-$dimensional small-world percolation network over n nodes, with probability at least $1 - \frac{1}{n^3}$ the NoN-GREEDY algorithm routes a message from s to t in $O(\frac{\log n}{\log\log n})$ hops. The probability is taken over the configuration of the graph.*

Note that the high probability bound of Theorem 4.2.2 implies that with high probability the NoN algorithm finds short paths between *all* pairs of nodes.

Proof of Theorem 4.2.2. The L_1 distance between any two nodes is at most n. So we assume the worst case - that the distance between the source and target is n. We partition the routing into two phases. In the first phase, the message is routed so that the remaining distance to the target diminishes to $e^{\sqrt{\log n}}$ or less. In the second phase, the message covers the remaining distance. We show that each phase takes $O(\log n/ \log\log n)$ w.h.p., thus proving the theorem. The first phase was handled in Lemma (6.1) from [33].

Lemma 4.2.3 ([33]). *If $m \geq c \log n/ \log\log n$, for some constant c which depends only on the dimension, then after m NoN-GREEDY routing steps, the message would reach a node that lies at distance $e^{\sqrt{\log n}}$ or less from the destination, with probability at least $1 - \frac{1}{n^3}$.*

The second phase of the routing could in fact be performed by plain GREEDY routing.

Lemma 4.2.4. *Given that the current location of the message from the source is at distance at most $e^{\sqrt{\log n}}$ from its destination, then with probability at least $1 - \frac{1}{n^3}$, the message would reach its destination within $O(\log n/ \log\log n)$ GREEDY steps.*

First we show the effect of a single a NoN hop:

Claim 4.2.5. *Let δ and δ' denote the distance from the destination before and after performing a single NoN* GREEDY *hop. There is an $\epsilon = \epsilon(d) > 0$ such that for any sequence of hops leading to the current node and for all $k > 0$*

$$\Pr[\delta' \leq \lceil (1 - \frac{1}{k})\delta \rceil] \geq 1 - \frac{1}{k^\epsilon}.$$

Proof. The proof will show that the Claim holds even for a single *greedy* hop. A single NoN hop is always longer than a single greedy hop. Assume the message is at node $\vec{0}$, and the target note t is such that $||t||_1 = \delta$. For each integer k define B_k to be all nodes with distance at most $(1 - \frac{1}{k})\delta$ from t (for notational convenience we remove the ceilings and floors). We calculate the probability there is an edge from $\vec{0}$ to the ball B_k. Define ℓ_i to be the number of vertices x such that $||x|| = i$ and x is in B_k. We have:

$$\Pr[\vec{0} \text{ is not connected to } B_k] = \prod_{i=\delta/k}^{\delta} (1-i^{-d})^{\ell_i} \leq \prod_{i=2\delta/k}^{\delta} (1-i^{-d})^{\ell_i} \leq \prod_{i=2\delta/k}^{\delta} e^{-\ell_i/i^d} = \exp(-\sum_{i=2\delta/k}^{\delta} \frac{\ell_i}{i^d})$$

Now assuming that ℓ_i is $\Theta(i^{d-1})$ for $\frac{2\delta}{k} \leq i \leq \delta$, for some constant ϵ it holds that

$$\exp(-\sum_{i=2\delta/k}^{n} \frac{\ell_i}{i^d}) \leq \exp(-\Theta(\sum_{i=2\delta/k}^{n} \frac{1}{i})) \leq \frac{1}{k^\epsilon}$$

which proves the claim. It remains to show that indeed $\ell_i = \Theta(i^{d-1})$ for $\frac{2\delta}{k} \leq i \leq \delta$. There are $\Theta(i^{d-1})$ nodes at distance i from $\vec{0}$, we need to show that a constant fraction of them are in B_k, i.e with distance at most $(1 - \frac{1}{k})\delta$ from t. Let i take some value $2\delta/k \leq i \leq \delta$ and let x be a point on a shortest path from $\vec{0}$ to t such that $||x|| = \lceil \frac{1}{2}(\delta/k + i) \rceil$. Note that $x \in B_k$, furthermore, all the points in distance $i - ||x|| = \lfloor \frac{1}{2}(i - \delta/k) \rfloor$ from x are also in B_k. Note that $\lfloor \frac{1}{2}(i - \delta/k) \rfloor$ is $\Theta(i)$ therefore there are $\Theta(i^{d-1})$ points at distance $\lfloor \frac{1}{2}(i - \delta/k) \rfloor$ from x. How many of them are of distance i from $\vec{0}$? All the points of equal distance from x are evenly divided between the 2^d quadrants of the ball around x. It follows that a $2^{-d} = \Theta(1)$ fraction of them are at distance i from $\vec{0}$, which concludes the proof of Claim 4.2.5. Figure 4.2 illustrates these calculations. □

Proof of Lemma 4.2.4. Claim 4.2.5 analyzed the case of a single NoN hop. Each hop (whether NoN or Greedy) examines edges towards the target, and eventually takes the edge (or two edges) which covers the longest distance. Therefore the portion of the graph that was encountered on previous hops is disjoint from the portion of the graph encountered in the current hop. In other words, the length of each hop is a random variable which depends only on the distance from the target and is independent from previous hops. Therefore we can use Claim 4.2.5 iteratively: set $k = \log^{1/4} n$, the probability the distance is reduced by a factor of $1 - \frac{1}{(\log n)^{1/4}}$ is $1 - \frac{1}{\log^\epsilon n}$. This means that $o(\log n / \log\log n)$ steps, each reduces the distance by $1 - \frac{1}{(\log n)^{1/4}}$, would route the message to the destination. We prove this occurs with probability $1 - \frac{1}{n^3}$ using the following argument: Let X_i be the random Bernoulli variable indicating whether the i^{th} NoN-hop have failed in reducing the

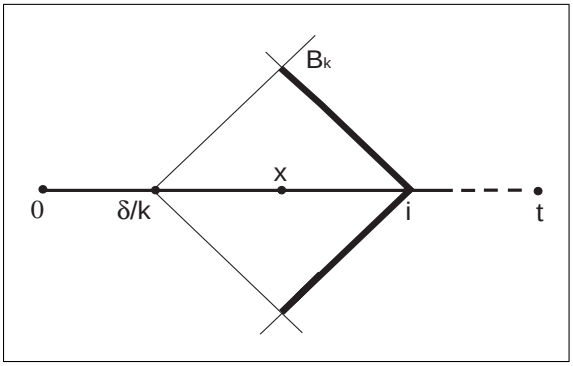


Figure 4.2: The bold line indicates points which are in B_k and are exactly distance i from $\vec{0}$.

distance by a factor of $1 - \frac{1}{(\log n)^{1/4}}$. We know that $\Pr[X_i = 1] \leq \frac{1}{\log^{\epsilon} n}$. Now assume that the variable X_i is simulated by tossing $\epsilon \log\log n$ fair coins and setting $X_i = 1$ if all coins turned up to be 1. Now we have $c \log n$ fair coins, and if less than $\frac{3}{4}$ of the coins turned up to be 1 the algorithm will not fail. The standard Chernoff bound [30] shows there is a constant c such that this happens with probability at least $1 - \frac{1}{n^3}$. □

The proof of Theorem 4.2.2 is now completed by combining Lemma 4.2.3 which handled the first phase of the routing, with Lemma 4.2.4 which handled the second phase of the routing. □

Do People Use the NoN-GREEDY Algorithm in Social Networks?

Since the original motivation of analyzing small-world graphs was the modeling of social networks, it is interesting to check whether people use the NoN-GREEDY algorithm when they navigate in a social network. Recently Dodds *et al* [37] repeated the famous experiment of Milgram [94] in which letters were passed between random nodes on a social network where edges corresponds to say, an acquaintance known by first name. In the Dodds *et al* experiment participants were given a target and were asked to forward an email to some person they were acquainted with. The goal of forwarding was to ensure that the email would reach its destination quickly. The participants were also asked to explain *why* they chose the person from among their set of acquaintances. It appears that in the first two steps of the "routing", which are most meaningful, about 25% of the people sent the message to a recipient for one of the following reasons:

1. The recipient was known to have traveled to the target's geographical region.
2. The recipient's family was known to have originated from the target's geographical region.

Both reasons suggest that the recipient received the message based on who his/her (possible) acquaintances were, and not on the individual characteristics of just the recipient. Other reasons, such as – "the recipient has the same education as the target" – could be viewed both as greedy and NoN-GREEDY steps. We can conclude that at least some of the time, the NoN-GREEDY algorithm was used.

4.3 NoN in Small-World P2P Networks

In this section, we analyze NoN-GREEDY routing for various randomized P2P routing networks which are related to the small world model and the small world percolation model discussed in the previous Section. Skip Graphs, which are of a different flavor, are analyzed in Section 4.4. We begin by defining these networks formally. For each of the following we assume there are $n = 2^\ell$ nodes arranged on a circle.

- **Randomized-Hypercube** [28, 53]: The out-degree of each node is ℓ. For each $1 \leq i \leq \ell$, 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. Edges are directed. Out-degree is $\ell = \log n$.
- **Randomized-Chord** [127, 53]: 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 $0 \leq i < \ell$, node $\mathbf{x}$ creates an edge with node $(\mathbf{x} + 2^i + r(i)) \bmod n$. Edges are directed. Each node has out-degree $\ell = \log n$.
- **Symphony** [87, 14]: Node $\mathbf{x}$ establishes a *short-distance* edge with node $(\mathbf{x}+1) \bmod n$. Node $\mathbf{x}$ also establishes $k \geq 1$ *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 $\lceil \mathbf{x} + r \rceil \bmod n$. Edges are directed. The resulting graph is thus a multi-graph node $\mathbf{x}$ could be connected to $\mathbf{y}$ by more than one edge. The out-degree of each node is $k + 1$.

Symphony with $k = 1$ is identical to Kleinberg's construction [70] in one dimension. Randomized-Hypercube and randomized-Chord are structurally similar to small-world percolation networks with $d = 1$ (see Definition 4.2.1). An important distinction is that the out-degree for each of the P2P routing networks is fixed.

Some easy adaptations of Lemma 4.2.3 and 4.2.4 could be used to prove the following theorem:

Theorem 4.3.1. *Given two nodes s, t in a randomized-Chord or a randomized-Hypercube network over n nodes, with probability at least $1 - \frac{1}{n^3}$ the NoN-GREEDY algorithm routes a message from s to t in $O(\frac{\log n}{\log \log n})$ hops, (the probability is taken over the configuration of the graph).*

The Theorem implies that these P2P networks are degree optimal using NoN. In Section 4.5 we show that GREEDY routing takes $\Omega(\log n)$, thus GREEDY makes suboptimal routing decisions.

In Symphony with out-degree $k + 1$ the expected path length found by GREEDY is $\Theta(\frac{\log^2 n}{k})$. The following Theorem shows that NoN-GREEDY improves upon GREEDY.

Theorem 4.3.2. *The expected number of hops taken by NoN-GREEDY to route between any two nodes in Symphony is $O\left(\frac{\log^2 n}{k \log k}\right)$, when $1 \leq k \leq \log n$ and the expectation is over the formation of the graph.*

Theorem 4.3.2 means that NoN improves upon GREEDY for any degree. When the degree is $\log n$ then NoN improves such that it is degree optimal w.h.p. Martel and Nguyen show that the *diameter* of symphony for any constant k is $O(\log n)$, which means that for these values of k NoN does not route along approximately shortest paths.

Proof. Consider node $\mathbf{x}$ that holds a message destined for node $\mathbf{y}$ lying clockwise distance d away. It is proven in [87] that GREEDY routing takes $O(\frac{\log n \log d}{k})$ hops. Therefore, if $\log d \leq \log n / \log k$,

then the remaining distance can be covered by NoN (which is faster than plain GREEDY) in $O(\log^2 n/(k \log k))$ hops.

We now consider large d satisfying $\frac{\log n}{\log k} < \log d \leq \log n$. Let $r(d) = \frac{ck \log d}{\log n}$ where d is the clockwise distance currently remaining and c is a constant that we will shortly fix. Since $\frac{\log n}{\log k} < d \leq \log n$, we deduce that $\frac{ck}{\log k} < r(d) \leq ck$.

Lemma 4.3.3. *Let $\mathcal{E}$ denote the event that the current node is able to diminish the remaining distance from d to at most $\frac{d}{r(d)}$ in (at most) two hops, then $\Pr[\mathcal{E}]$ is $\Omega(\frac{k}{\log n})$, independent of d.*

Thus the expected number of nodes encountered before event $\mathcal{E}$ occurs is $O(\frac{\log n}{k})$. Since $\frac{ck}{\log k} < r(d)$, there can be at most $O(\frac{\log n}{\log k})$ such events for a total of $O(\frac{\log^2 n}{(k \log k)})$ hops. When d becomes small enough to satisfy $\log d < \frac{\log n}{\log k}$, plain GREEDY routing will take at most $O(\frac{\log^2 n}{k \log k})$ hops. Summing the two, the total number of hops is $O(\frac{\log^2 n}{k \log k})$. Thus it only remains to prove Lemma 4.3.3.

Proof that $\Pr[\mathcal{E}]$ is $\Omega(\frac{k}{\log n})$: Denote by $B(\mathbf{x})$ the number of nodes connected by an edge to $\mathbf{x}$ which are at a clockwise distance of at most d away. By the definition of Symphony it holds that $\mathbb{E}[B(\mathbf{x})] \geq \frac{k \log d}{\log n}$. Let $d' = \lceil d(1 - \frac{1}{r(d)}) \rceil$. Let ψ denote the event that such a node has a link in clockwise distance $[d', d]$ from $\mathbf{x}$. Since ψ is independent from $B(\mathbf{x})$ and $\Pr[\mathcal{E}]$ is monotone in $B(\mathbf{x})$ we have that overall, the probability that one or more of these nodes has a link in distance $[d', d]$ from $\mathbf{x}$ is:

$$\Pr[\mathcal{E}] \geq 1 - (1 - \Pr[\psi])^{\frac{k \log d}{2 \log n}} \cdot \Pr\left[B(\mathbf{x}) \geq \frac{k \log d}{2 \log n}\right] \tag{4.1}$$

We handle each element in (4.1) separately. $B(\mathbf{x})$ is the sum of k Bernoulli variables, each with success probability $\frac{\log d}{\log n} \geq \frac{1}{k}$. By Chernoff's bound (e.g [55]) we have:

$$\Pr\left[B(\mathbf{x}) \leq \frac{k \log d}{2 \log n}\right] \leq \exp(-\frac{1}{8}\mathbb{E}^2[B(\mathbf{x})]) \leq e^{-\frac{1}{8}}$$

Next we show that $\Pr[\psi]$ is $\Omega(\frac{k}{r(d) \log n})$. Recall that ψ denotes the probability that node $\mathbf{x}$ or one of its neighbors has a link in distance $[d', d]$ where $d' = \lceil d(1 - \frac{1}{r(d)}) \rceil$. According the definition of Symphony, the probability node $\mathbf{x}$ or one of its neighbors does not have a link in distance $[d', d]$ is

$$\Pr[\overline{\psi}] \leq \left(1 - \frac{\log(\frac{r(d)}{r(d)-1})}{\log n}\right)^k \leq \left(1 - \frac{\log(1 + \frac{1}{r(d)})}{\log n}\right)^k \leq \left(1 - \frac{1}{2r(d) \log n}\right)^k \leq e^{-\frac{k}{2r(d) \log n}}$$

In the third inequality we used the fact that $\log(1 + \frac{1}{r(d)}) \leq \frac{1}{2r(d)}$. We have:

$$\Pr[\psi] \geq 1 - e^{-\frac{k}{2r(d) \log n}} \geq \frac{c'k}{r(d) \log n}$$

for some constant c'. Now all that remains is to substitute in (4.1). We had defined $r(d) = \frac{ck \log d}{\log n}$. We set $c \geq c'$ which ensures that

$$\frac{c'k}{r(d) \log n} \cdot \frac{k \log d}{\log n} \leq \frac{k}{\log n} \leq 1.$$

Using the fact that $1-(1-x)^t \geq xt/2$ if $x \in (0,1)$ and $xt \leq 1$, we deduce that

$$\Pr[\mathcal{E}] \geq \frac{c'k^2 \log d}{2r(d)\log^2 n}.$$

Substituting $r(d) = \frac{c'k\log d}{\log n}$, we get $\Pr[\mathcal{E}]$ is $\Omega(\frac{k}{\log n})$ as needed. This completes the proof of Lemma 4.3.3 and thus of Theorem 4.3.2. □

4.4 NoN in Skip Graphs

In this section, we analyze NoN-GREEDY routing in skip-graphs [16] and SkipNets [57], which adapt skip-lists [112] for creating a randomized P2P routing network[2]. We follow the description in [16].

4.4.1 A Review of Skip Graphs

In a skip graph, each node represents a resource to be searched. Node x holds two fields: the first is a key, which is arbitrary and may be the resource name. Nodes are ordered according to their keys. We assume for notational convenience that the keys are the integers $1, 2, \ldots, n$; as the keys have no function in the construction other than to provide an ordering and a target for searches there is no loss of generality. The second field is a membership vector $m(x)$ which is for convenience treated as an infinite string of random bits chosen independently by each node; in practice, it is enough to generate an $O(\log n)$-bit prefix of this string with overwhelming probability.

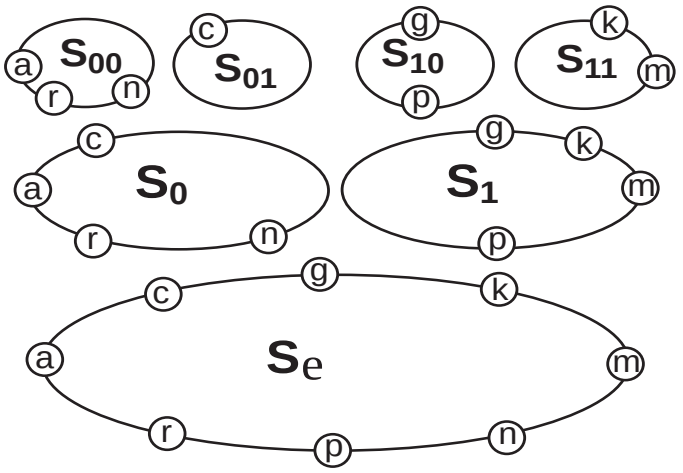
The nodes are ordered lexicographically by their keys in a circular doubly-linked list S_ϵ so that node i is connected to $i-1 \mod n$ and $i+1 \mod n$. For each finite bit-vector σ, an additional circular doubly-linked list S_σ is constructed by taking all nodes whose membership vectors have σ as a prefix, and linking adjacent nodes in the lexicographic key order. More formally, let $m(x)_k$ be the restriction of $m(x)$ to its first k bits; then nodes $x < y$ are connected by an edge if there exists some k such that $m(x)_k = m(y)_k$, and either (a) $m(z)_k \neq m(x)_k$ for each z between x and y, or (b) $m(z)_k \neq m(x)_k$ for all $z > x$ and all $z < y$. In such case we say the edge (x, y) *corresponds* to a prefix of length k. Note that the cycle edges could be seen as corresponding to the empty prefix.

In analyzing a skip graph as a graph, we treat each pair of links as a single undirected edge, and take the union of the resulting edge sets for all lists S_σ. An example of a skip graph is depicted if Figure 4.3

The skip graph data structure is well fit for implementation in a dynamic setting. It is shown in [16] and [57] that the joining and leaving of a node takes $O(\log n)$ time and $O(\log n)$ messages. Efficient search algorithms and repair algorithms are also shown, so in this respect skip graphs are comparable to other constructions of DHT's. The main merit of skip-graphs is the following: edges do not depend on the keys themselves but rather on their ordering and the random vectors. Thus the keys may be arbitrary and can carry *semantic meaning*. Furthermore, since the nodes are ordered by their keys, the skip-graph data structure supports *prefix search*. This is in stark contrast with the other networks we discuss, which require that keys be random.

Claim 4.4.1. *Let x, y be two nodes such that $x < y$. The probability there exists a (clockwise) edge (y, x) is $\Theta(\frac{1}{y-x})$.*

[2]There are some differences between the two suggestions, but essentially they are the same, and our results apply for both.



node	vector
a	001...
c	011...
g	100...
k	110...
m	111...
n	000...
p	101...
r	000...

Figure 4.3: An example of a skip graph.
Only three levels are displayed. The actual graph is the union of all cycles.

Proof. Denote by $y \sim x$ the event that (y, x) is an edge. The probability that $m(x)$ and (y) share the same length k prefix is 2^{-k}. The probability none of the nodes $x+1, x+2, \ldots, y-1$ have this prefix is $(1-2^{-k})^{|x-y|-1}$. Since these two events are independent, the probability (x, y) is an edge which corresponds to a prefix of length k is $2^{-k} \cdot (1-2^{-k})^{|x-y|-1}$. Setting $k = \lfloor \log(y-x) \rfloor$ we have $\Pr[y \sim x] \in \Omega(\frac{1}{y-x})$. On the other hand $\Pr[y \sim x] \leq \sum_k 2^{-k} \cdot (1-2^{-k})^{(y-x)-1} \in O\left(\frac{1}{y-x}\right)$ □

The Claim above demonstrates the structural similarity skip graphs share with small world percolation graphs - thus the effectiveness of NoN on skip graphs is perhaps not that surprising. The claim above implies that the expected degree of each node is about $\log n$. It is easy to see that w.h.p. the maximum degree in the graph is logarithmic: With high probability all prefixes of length $3 \log n$ are different, therefore all the edges in the graph correspond to prefixes of length at most $3 \log n$.

4.4.2 Routing in Skip Graphs

The routing algorithm suggested in [16], [57] routes the message using the longest prefix of $m(x)$ possible, without overshooting the target. In other words it is a greedy algorithm which moves the message as close to the target as possible without overshooting, and routes in $O(\log n)$ hops on expectation[3]. We improve this routing by showing in Theorem 4.4.2 that the NoN-Greedy algorithm routes in $O(\log n / \log \log n)$ hops w.h.p, and by showing in Theorem 4.5.1 that the GREEDY algorithm needs $\Omega(\log n)$ time to route. Each NoN hop considers paths of two edges and routes the message as close as possible to the target without overshooting it. That is, assuming the target node is 0, at each hop the algorithm routes the message to the neighbor of neighbor with the smallest positive id[4]. Since the target node is set to 0 we abuse notation slightly and let a node's

[3] A high probability bound can be easily derived using the machinery introduced in this section.
[4] For notational convenience we assume that the routing is always done in the clockwise direction.

key indicate its distance from the target.

Theorem 4.4.2. *Let s be some node in a skip graph of n nodes, with probability $1 - \frac{1}{n^2}$ the NoN algorithm finds a path between node s and node 0 of length $O(\frac{\log n}{\log \log n})$, where the probability is taken over the choices of membership vectors.*

Given the low probability of failure, Theorem 4.4.2 implies that with high probability *all* nodes are connected to node 0 via a short path:

Corollary 4.4.3. *With probability at least $1 - \frac{1}{n}$ the diameter of a skip-graph with n nodes is $O(\frac{\log n}{\log \log n})$, where the probability is taken over the choices of membership vectors.*

The proof of Theorem 4.4.2 is rather technical, though the outline is similar to that of percolation small world graphs. The proof is divided into two parts. In the first we show that with high probability the message reaches quickly a distance of $\exp(\sqrt{\log n})$ from the target and in the second part we show that with high probability all nodes of distance at most $\exp(\sqrt{\log n})$ are connected to the target through short chains. We need to re-prove Lemmas 4.2.3 and 4.2.4 and deal with the dependencies created by the properties of skip graphs.

The First Phase of the Routing

Let X_m be the point the NoN algorithm reached after performing m NoN hops. Since the target is 0, X_m also represents the distance from the target. $X_0 = s$ is the starting point. The following is a restatement of Lemma 4.2.3 for the case of skip graphs.

Lemma 4.4.4. *There exists a constant c such that for $m \geq c(\frac{\log n}{\log \log n})$ with probability at least $1 - \frac{1}{n^2}$ it holds that $X_m \leq \exp(\sqrt{\log n})$.*

The general outline of the proof follows that of Lemma 4.2.3 in [33]. Our goal is to show that each NoN hop the distance to the target reduces by some poly-logarithmic factor. The main source of technical difficulty in skip graphs is that the probability of a hop of a certain length depends upon all the hops taken so far. In other words the value $\Pr[X_r \leq \frac{X_{r-1}}{(\log n)^{1/4}}]$ is now a *random variable* whose distribution depends upon r and the membership vectors sampled in the segment $[s, X_{r-1}]$. Fix some $1 \leq r \leq m$. It is sufficient to prove the following lemma:

Lemma 4.4.5. *Denote by $\mathcal{E}_r$ the event that $\Pr[X_r \leq \frac{X_{r-1}}{(\log n)^{1/4}}] \geq 1 - \frac{c}{\sqrt{\log n}}$. There exists a constant c such that $Pr[\mathcal{E}_r] \geq 1 - \frac{1}{n^3}$*

Before proving Lemma 4.4.5 lets see why it derives Lemma 4.4.4. With probability $1 - \frac{1}{n^3}$ the event $\mathcal{E}_r$ holds, so with probability greater than $1 - \frac{1}{n^2}$ the event $\mathcal{E}_r$ holds for every r, i.e. all through the path. Call a NoN hop *successful* if it reduces the distance to 0 by a factor of $(\log n)^{-1/4}$. It holds that $\frac{4 \log n}{\log \log n}$ successful hops suffice to bring the message to a distance of $\exp(\sqrt{\log n})$. The previous discussion implies that with high probability each NoN hop along the path is not successful with probability at most $c \log^{-1/2} n$. Therefore we can simulate the process in the following way: for each hop we toss $\frac{1}{2} \log \log n + \log c$ *fair* coins, and the hop fails if *all* of them turn out successful. As seen in the proof of Lemma 4.2.4, for large enough m, with probability $1 - \frac{1}{n^2}$ there would be $\frac{4 \log n}{\log \log n}$ successful NoN hops within the first $\frac{m \log n}{\log \log n}$ attempts.

Proof of Lemma 4.4.5. Our goal is to use the same outline as in [33]: Assume the current node is x (i.e. $X_{r-1} = x$). The proof has two components. First we show that with sufficiently high probability x has $\Omega(\log x)$ neighbors in $[x-1, 0]$, then we show that with sufficiently high probability *one* of these neighbors is part of a successful NoN step.

Before we do that however we need to build the machinery that will help us deal with the dependencies. The main problem is that conditioning on the path taken so far, the membership vectors in the segment $[x-1, 0]$ are not drawn from the uniform distribution. To see this consider for instance the case that for some $y \in [x-1, 0]$ it holds that $m(y)_k = m(s)_k$ for large k, then the edge (s, y) would exist and the NoN path starting at s would bypass x altogether. In other words the conditioning that $X_r = x$ affects the distribution of membership vectors in the segment $[x-1, 0]$. We conclude that for every choice of membership vectors in the segment $[n, x]$ there is a set of vectors $\mathcal{F}_x \subseteq \{0,1\}^*$ such that the following two conditions hold:

i If a node $y \in [x-1, 0]$ has $m(y) \in \mathcal{F}_x$ then there would be an edge from some node in $[n, x+1]$ to y that would cause the NoN path from s to 0 to bypass node x.

ii If all nodes in $[x-1, 0]$ have their membership vectors drawn from $\{0,1\}^* \setminus \mathcal{F}_x$ then node x would belong to the NoN path starting from s.

It holds therefore that when conditioning on the path taken so far, the membership vectors in $[x-1, 0]$ are drawn *uniformly* and *independently* from $\{0,1\}^* \setminus \mathcal{F}_x$.

Denote by $\mu(\mathcal{F}_x)$ the measure of $\mathcal{F}_x$, i.e. the probability a random membership vector falls within $\mathcal{F}_x$. Note that every choice of membership vectors for the nodes $[n, x]$ defines a path to x and a set $\mathcal{F}_x$, so $\mu(\mathcal{F}_x)$ is a random variable determined by the membership vectors in $[n, x]$. Our goal now is to show that with high probability $\mu(\mathcal{F}_x)$ is small. For every $0 \leq \alpha \leq 1$ we have:

$$\Pr[\mu(\mathcal{F}_x) \geq \alpha \mid X_{r-1} = x] \leq \frac{\Pr[X_{r-1} = x \mid \mu(\mathcal{F}_x) \geq \alpha]}{\Pr[X_{r-1} = x]}.$$

For $\{X_{r-1} = x\}$ to occur all vectors in $[0, x-1]$ must be outside $\mathcal{F}_x$, therefore for every $0 \leq \alpha \leq 1$ it holds:

$$\Pr[X_{r-1} = x \,|\, \mu(\mathcal{F}_x) \geq \alpha] \leq (1-\alpha)^x$$

Let $S_{r-1} \subseteq [0, n]$ denote the set of nodes such that $\Pr[X_{r-1} = x] \geq \frac{1}{n^3}$. The event that $x \notin S_{r-1}$ is negligible. For every $x \in S_{r-1}$ we have:

$$\Pr[\mu(\mathcal{F}_x) > \alpha \mid X_{r-1} = x] \leq n^3(1-\alpha)^x \tag{4.2}$$

Setting $\alpha = \frac{4 \log n}{x}$ we have that with probability $\geq 1 - \frac{1}{n^3}$, it holds that $\mu(\mathcal{F}_{X_{r-1}}) \leq \frac{4 \log n}{x}$. Denote by A the high probability event that $x \in S_{r-1}$ and $\mu(\mathcal{F}_{X_{r-1}}) \leq \frac{4 \log n}{x}$.

Now all that remains is to show that the occurrence of A implies the occurrence of $\mathcal{E}_r$. First we show that the occurrence of A implies that the distribution of membership vectors in $[x-1, 0]$ is almost uniform.

For a prefix $\psi \in \{0,1\}^k$ denote by b_ψ the probability a random and uniform vector in $\{0,1\}^*$ falls in $\mathcal{F}_x$ conditioned on its prefix being ψ (i.e. if u is sampled uniformly from $\{0,1\}^*$ then $b_\psi = \Pr[u \in \mathcal{F}_x | m(u)_k = \psi]$). Denote by w_ψ the probability a random vector in $\{0,1\}^* \setminus \mathcal{F}_x$ has ψ as a prefix, i.e. w_ψ is the probability ψ is a prefix in the conditional distribution of vectors in $[x-1, 0]$.

Claim 4.4.6. *For every integer $k > 0$*

$$\sum_{\psi \in \{0,1\}^k} b_\psi = \mu(\mathcal{F}_x) \cdot 2^k \tag{4.3}$$

$$\frac{1 - b_\psi}{2^k} \leq w_\psi \leq \frac{1}{2^k(1 - \mu(\mathcal{F}_x))} \tag{4.4}$$

Proof. Let u be a vector sampled uniformly in $\{0,1\}^*$. Equation (4.3) follows since:

$$\mu(\mathcal{F}_x) = \Pr[u \in \mathcal{F}_x] = \sum_{\psi \in \{0,1\}^k} \Pr[u \in \mathcal{F}_x \mid m(u)_k = \psi] \Pr[m(u)_k = \psi] = 2^{-k} \sum_{\psi \in \{0,1\}^k} b_\psi$$

The first inequality in Equation (4.4) follows since $\frac{1-b_\psi}{2^k}$ is the probability a single sample from $\{0,1\}^*$ has ψ as a prefix and falls outside $\mathcal{F}_x$. The second inequality holds since $\frac{1}{2^k(1-\mu(\mathcal{F}))}$ is the normalized probability after removing a measure $\mu(\mathcal{F}_x)$. □

We are now set to prove the claims needed for the proof of Lemma 4.4.5.

Lemma 4.4.7. *Let $B(x)$ be the number of nodes in $[0, x-1]$ which are connected to x by an edge corresponding to a prefix of length at most $\frac{1}{10}\log x$.*

$$\Pr\left[B(x) \geq \frac{1}{100}\log x \,\middle|\, A\right] \geq 1 - \frac{1}{\sqrt{\log n}}$$

Proof. First note that for every $k \leq \log(\frac{1}{\mu(\mathcal{F}_x)}) - 2$ Claim 4.4.6 implies that $\sum b_\psi \leq \frac{1}{4}$ and therefore for every $\psi \in \{0,1\}^k$ it holds that

$$\frac{3}{4} \cdot \frac{1}{2^k} \leq w_\psi \leq \frac{5}{4} \cdot \frac{1}{2^k}$$

Next we claim that with high probability there is a series of nodes $y_1 > y_2 > \ldots > y_{1/10 \log x}$ where y_k is the largest node in $[x-1, 0]$ such that $m(y_k)_k = m(x)_k$ for $1 \leq k \leq \frac{1}{10}\log x$. We toss the membership vectors in $[x-1, 0]$ one by one from $m(x-1)$ to $m(0)$ finding the y_i's one by one. In other words - consider a series of independent geometric random variables $g_1, g_2, \ldots$ where the parameter of g_i is the probability a vector shares a prefix of length i with $m(x)$. Since we conditioned on $\mu(\mathcal{F})$ being small, by the previous discussion the success probability of g_i is at least $\frac{3}{4}2^{-i}$. Each g_i is repeatedly tossed until there is a success, in which case g_{i+1} is tossed and so on. On expectation, the number of attempts needed until $g_{1/10 \log x}$ succeeds is at most $\sum_{i=1}^{1/10 \log x} 2^{i+1} \leq 4x^{1/10}$. We have x tosses at our disposal so by Markov's inequality, with probability greater than $1 - 4x^{-\frac{9}{10}}$ there is a series of nodes $y_1, y_2, \ldots, y_{\frac{1}{10}\log x}$ such that $m(x)_k = m(y_k)_k$. Typically it is not the case that all these nodes are neighbors of x. If for some $j < i$ it holds that $m(x)_i = m(y_j)_i$ then y_i is not a neighbor of x. It holds however that

$$\Pr[m(y)_{i+1} = m(x)_{i+1} | m(y)_i = m(x)_i] \leq \frac{\frac{3}{4}2^{-(i+1)}}{\frac{5}{4}2^{-i}} = \frac{5}{6}$$

It follows then that each y_i is a neighbor of x with probability at least $\frac{1}{6}$ and independently from all other y_i's. So on expectation at most $\frac{1}{6}$ of the y_i's are neighbors of x. By Chernoff's bound, the probability less than $\frac{1}{10}$ of the y_i's are connected to x is at most $e^{-\epsilon \log x}$, where ϵ is some constant.

Now, since $\log x \geq \sqrt{\log n}$ we conclude that with probability greater than $1 - e^{\epsilon\sqrt{\log n}} > 1 - \frac{1}{\sqrt{\log n}}$ it holds that $B(x) \geq \frac{1}{100}\log x$. □

We continue with the proof of Lemma 4.4.5. recall that $y_1, y_2, \ldots, y_m$ are neighbors of x where $m \in \Omega(\log x)$ and each y_i corresponds to a prefix of length at most $\frac{1}{10}\log x$. It remains to show that with probability $1 - \frac{1}{\sqrt{\log n}}$ one of them has an edge towards $[\frac{x}{\log^{1/4} n}, 0]$. Consider the first $\log x$ bits of each $m(y_i)$.

Claim 4.4.8. *Let z be some node in $[x-1, 0]$. $\Pr[y_i \sim z \mid A] \geq \frac{1}{40x}$ where the edge corresponds to a prefix of length $\lfloor \log x \rfloor$.*

Proof. Let $L \subset \{0,1\}^{\log x}$ be vectors with $m(y_i)_i$ as a prefix, so $|L| \geq x^{9/10}$. Now, according to Claim 4.4.6 there are at least $x^{9/10} - 4\log^2 n$ prefixes $\psi \in L$ such that $\frac{1}{2x} \leq w_\psi \leq \frac{2}{x}$. Denote by y_ψ the probability that $m(y_i)_{\log x} = \psi$. The same arguments show that for the vast majority of vectors in L it holds that $\frac{1}{2x^{9/10}} \leq y_\psi \leq \frac{2}{x^{9/10}}$. So to conclude, in at least half of the prefixes in L the conditioning on A changes the probability by a factor of at most 2. Now:

$$\Pr[y_i \sim z \mid A] \geq \sum_{\psi \in L} y_\psi w_\psi (1 - w_\psi)^x \geq \frac{1}{40x}$$

□

Claim 4.4.8 implies that for every node $z \in [\frac{x}{\log^{1/4} n}, 0]$, the probability (y_i, z) is an edge corresponding to a prefix of length $\lfloor \log x \rfloor$ is $\Theta(1/x)$. Let Y_i be the random variable indicating that y_i is connected to some node in$[\frac{x}{\log^{1/4} n}, 0]$ with an edge which corresponds to a prefix of length *exactly* $\lfloor \log x \rfloor$. There could be at most one such edge so

$$\Pr[Y_i] \geq \frac{1}{40\log^{1/4} n}$$

For $i \neq j$ it holds that $\Pr[Y_i|Y_j] \leq \Pr[Y_i]$. The reason is that if Y_j holds then $m(y_j)_{\log x}$ appears in $[\frac{x}{\log^{1/4} n}, 0]$ and does not appear in $[x, \frac{x}{\log^{1/4} n}]$, both events reduce the probability of Y_i, and this holds also when conditioning on A. In other words, the $Y_i|A$ are negatively correlated, so:

$$\Pr[\overline{Y_1} \wedge \ldots \wedge \overline{Y_m} \mid A] \leq \prod_{i=1}^{\Theta(\log x)} \left(1 - \Theta(\log^{-1/4} n)\right) \leq \exp(-\Theta(\log^{1/4} n)) \leq \frac{1}{\sqrt{\log n}}$$

where in the second inequality we used the assumption that $\log x \geq \sqrt{\log n}$. Adding all the error probabilities implies that given the high probability event A, the event $\mathcal{E}_r$ holds, and as seen, Lemma 4.4.5 implies Lemma 4.4.4.

□

Routing the Remaining Distance

For the second phase of the routing we basically re-prove Lemma 4.4.4 with different number crunching. Assume the Skip Graph contains $e^{\sqrt{n}}$ nodes and as before denote by X_i the location of the NoN algorithm after i steps.

Lemma 4.4.9. *There exists a constant c such that for $m \geq c(\frac{\log n}{\log\log n})$ with probability at least $1 - \frac{1}{n^2}$ it holds that $X_m \leq c(\frac{\log n}{\log\log n})$.*

Proof. We use the same notation and mechanism of Lemma 4.4.4. Assume $X_{r-1} = x$ is the current node. Note that $x-1$ is always a neighbor of x, so it is enough to show that a greedy choice from $x-1$ would reduce the distance by a factor of $1 - \frac{1}{\log^{1/4} n}$ with probability $1 - \frac{1}{\log^{\epsilon} n}$ for some $\epsilon \geq 0$.

Recall that Equation (4.2) states that $\Pr[\mu(\mathcal{F}_x) > \alpha \mid X_{r-1} = x] \leq n^3(1-\alpha)^x$. The Equation implies that for every δ there exist $c(\delta)$ such that if $x \geq \frac{c\log n}{\log\log n}$, then $\Pr[\mu(\mathcal{F}_x) > 1 - \log^{-\delta} n \mid X_{r-1} = x] \leq \frac{1}{n^3}$. The exact value of δ would be set later on. As before we name this high probability event A and condition on it to hold.

Denote by $k = 0.8 \log x$. If k is not integer then take a coefficient slightly smaller than 0.8. As before, for every prefix ψ of length k denote by w_ψ the probability that ψ is sampled condition on the path taken so far. Now:

$$\begin{aligned}
\Pr\left[X_r \leq (1 - \frac{1}{\log^{1/4} n})x \mid A\right] &\geq \Pr\left[x - 1 \sim y \text{ for some } y \leq (1 - \frac{1}{\log^{1/4} n})x\right] \\
&\geq \sum_{\psi \in \{0,1\}^k} w_\psi (1-w_\psi)^{\frac{x}{\log^{1/4} n}} \left(1 - (1-w_\psi)^{(1 - \frac{1}{\log^{1/4} n})}\right) \\
&= \sum_{\psi \in \{0,1\}^k} w_\psi (1-w_\psi)^{\frac{x}{\log^{1/4} n}} - \sum_{\psi \in \{0,1\}^k} w_\psi (1-w_\psi)^x \qquad (4.5)
\end{aligned}$$

Our goal is to show that the sum in Equation (4.5) is at least $1 - \frac{1}{\log^{\epsilon} n}$. Claim 4.4.6 implies that if A occurs then for every $\psi \in \{0,1\}^k$ it holds that $w_\psi \leq \frac{\log^{\delta} n}{x^{0.8}}$. We deal with the two sums of Equation (4.5) separately:

$$\begin{aligned}
\sum_{\psi \in \{0,1\}^k} w_\psi (1-w_\psi)^{\frac{x}{\log^{1/4} n}} &\geq \min\{(1-w_\psi)^{\frac{x}{\log^{1/4} n}}\} \\
&\geq \left(1 - \frac{\log^{\delta} n}{x^{0.8}}\right)^{\frac{x}{\log^{1/4} n}} \\
&\geq 1 - \frac{1}{\log^{1/25} n} \quad \text{for sufficiently small } \delta
\end{aligned}$$

The first inequality holds since $\sum w_\psi = 1$. In the last inequality we used the assumption that $x > \frac{\log n}{\log\log n}$.

For the second part of Equation (4.5) we divide the sum into elements of small and big weight. Denote by S all the elements $\psi \in \{0,1\}^k$ such that $w_\psi \leq \frac{1}{x^{0.9}}$. There are at most $x^{0.8}$ elements in S therefore

$$\sum_{\psi \in S} w_\psi (1-w_\psi)^x \leq \frac{x^{0.8}}{x^{0.9}} \leq \frac{1}{\log^{1/25} n}.$$

On the other hand

$$\sum_{\psi \notin S} w_\psi (1-w_\psi)^x \leq \max_{\psi \notin S}\{e^{-w_\psi x}\} \leq e^{-\frac{x}{x^{0.9}}} \leq \frac{1}{\log^{1/25} n}.$$

Now as before we have that $O(\frac{\log n}{\log\log n})$ distance reductions of factor $1-\frac{1}{\log^{1/4} n}$ are enough to bring the path to a distance of $O(\frac{\log n}{\log\log n})$ from the target, and with probability greater than $1-\frac{1}{n^2}$ this indeed happens within $O(\log n/\log\log n)$ attempts.

□

The remaining distance to the target could be covered by the cycle edges. Now the proof of Theorem 4.4.2 is concluded by union bounding the error probabilities of Lemmata 4.4.4 and 4.4.9.

4.5 Lower Bounds

In this section we prove that in order to find a path between nodes at distance n, a routing algorithm must either run in $\Omega(\log n)$ time w.h.p (i.e. $\Omega(\log n)$ hops), or must use additional knowledge about the neighbor's neighbors of a node. The lower bound holds for a model which generalizes the GREEDY algorithm, thus it applies for a larger family of algorithms which includes GREEDY. It holds both for small-world percolation networks and skip graphs.

A logarithmic lower bound of $\Omega(\log^2 n)$ for GREEDY routing in Kleinberg's construction [70] in one dimension was proved by Barrière *et al* [19]. Aspnes *et al* [14] extended the result to a larger family of random graphs. They show that if the average degree is $O(\log n)$ then GREEDY routing would take $\Omega(\log n)$ hops on average. The proof however is limited to the case where the nodes are set on a one dimensional line and the probability upon the edges has some symmetry assumptions that do not apply to skip graphs. We show lower bounds for small-world percolation networks and skip-graphs. Randomized-Chord, randomized-hypercube and Symphony are quite similar to small-world percolation networks, and the proofs could be adapted for each of them.

4.5.1 A Probing Model

Assume that our goal is to find a path between two specific vertices distance n apart, say node 0 and node n. In order to do so, an algorithm must *probe* the vertices of the graph, where the probing of a vertex reveals all the edges connected to it. Our lower bounds apply in a *probing* model, where we bound the number of probes needed to find a path. Clearly, a lower bound on the number of probes needed by the algorithm is a lower bound on the (sequential) time complexity of a routing algorithm.

We define a 1−local algorithm to be a probing algorithm with the following properties:

1. The algorithm begins by probing the node 0.
2. The algorithm only probes nodes to which it has already established a path from 0.

The term *local* derives from the assumption that the algorithm starts at 0 and is only allowed to probe nodes it has already reached. The term 1−local is used, since the probing of a node reveals its neighborhood of radius 1, i.e. its neighbors. If it is assumed that a probe reveals a neighborhood of radius k then the algorithm is termed *k-local*. Every routing algorithm which relies on local information only, corresponds to a 1−local probing algorithm. The GREEDY algorithm therefore is 1−local. The NoN-GREEDY algorithm could be viewed, following Theorems 4.2.2 and 4.4.2 as either a 2−local algorithm with $O(\log n/\log\log n)$ probes w.h.p, or as a 1−local algorithm having probing complexity of $O(\log^2 n/\log\log n)$. Other 1−local algorithms could be though of, see for instance [74].

4.5.2 Lower Bounds in the Probing Model

Theorem 4.5.1. *(i) In a skip graph - any* 1-local *algorithm that outputs a path between two nodes at distance n, must probe $\Omega(\log n)$ probes, with probability at least $1-\frac{1}{n^\epsilon}$. In particular, the expected number of probes is $\Omega(\log n)$.*
(ii) In a $d-$dimensional small-world percolation network - any 1-local *algorithm that outputs a path between two nodes at distance n, must probe $\Omega(\log n)$ probes, with probability at least $1-\frac{1}{n^\epsilon}$. In particular, the expected number of probes is $\Omega(\log n)$.*

The theorem implies that if a node holds only its neighbors then *any* routing algorithm would need $\Omega(\log n)$ probes w.h.p. Thus the assumption that nodes have some knowledge of their neighbor's neighbors is essential.

We first argue that GREEDY dominates any other $1-$local algorithm. The following lemma holds both for skip graphs and small-world percolation networks.

Lemma 4.5.2. *Let $\mathcal{A}$ be a $1-$local algorithm . Denote by A_d, G_d the random variables representing the number of probes it takes the algorithm $\mathcal{A}$ and the* GREEDY *algorithm respectively, to find a path between two nodes at distance d. For all $d > k > 0$ it holds that* $\Pr[G_d \leq k] \geq \Pr[A_d \leq k]$.

Proof. We distinguish between the two cases.

Small-World Percolation Networks For convenience, we label the target node as $\vec{0}$, and assume that the mesh is infinite. The trick is to give $\mathcal{A}$ some extra power. Assume that at some step, the closest node to $\vec{0}$ which $\mathcal{A}$ had found is at distance d from $\vec{0}$, where the distance is measured by the L_1 norm. At this point, we grant $\mathcal{A}$ access to all nodes outside a ball of radius d from $\vec{0}$. Now if $d_1 > d_2$ then for every configuration of edges, every move A can do in case the distance is d_1, is also available when the distance is d_2, so without loss of generality, for every k, $\Pr[A_{d_1} \leq k] \leq \Pr[A_{d_2} \leq k]$. In other words, for every k, $\Pr[A_d \leq k]$ is monotonically decreasing in d. The algorithm $\mathcal{A}$ samples some point v. The greedy choice is to sample a point closest 0, call that point u. Let $f(v)$ denote the the distance from $\vec{0}$ of the neighbor of v which is closest to $\vec{0}$. Now

$$\Pr[A_d \leq k] = \sum_{i<d} \Pr[f(v) = i] \cdot \Pr[A_i \leq k-1]$$

Since $||u|| \leq ||v||$ it holds that for every i, $\sum_{j=0}^{i} \Pr[f(u) = j] \geq \sum_{j=0}^{i} \Pr[f(v) = j]$. Since $\Pr[A_d \leq k]$ is monotonically decreasing we have:

$$\sum_i \Pr[f(u) = i] \Pr[A_i \leq k-1] \geq \sum_i \Pr[f(v) = i] \Pr[A_i \leq k-1]$$

In other words, the best thing $\mathcal{A}$ can do is sample the greedy point, which implies that the GREEDY algorithm dominates any other $1-$local algorithm.

Skip Graphs The technique used in the previous section applies here as well. Now if at some step the closest node to 0 which $\mathcal{A}$ had found is at distance d from 0 we supply to $\mathcal{A}$ both the access and the membership vectors of all the nodes in the segment $[n, d]$. We need to handle some dependencies. Denote by M_d the membership vectors of this segment. Assume that $\mathcal{A}$ probes point

v and GREEDY probes point u. Using the notation of the previous section we have

$$\Pr[(A_d \leq k)|M_d] = \sum_{i=0}^{d-1} \Pr[(f(v) = i)|M_d] \cdot \Pr[(A_i \leq k-1)|M_d]$$

It is easy to see that for every instance of M_d, it holds that $\Pr[f(v) = i|M_d] \leq \Pr[f(u) = i|M_d]$. To see this consider prefixes of length k. If v does not have a neighbor corresponding to a prefix of length k within the segment $[n, d]$ then the probability $f(v) = i$ due to a prefix of length k is equal to the probability $f(u) = i$ due to a prefix of length k. If v does have a neighbor corresponding to a prefix of length k in $[n, d]$ then $\Pr[f(v) = i] < \Pr[f(u) = i]$. Conclude that

$$\begin{aligned}\Pr[(A_d \leq k)|M_d] &= \sum_{i=0}^{d-1} \Pr[(f(v) = i) \,|\, M_d] \cdot \Pr[(A_i \leq k-1) \,|\, M_i] \\ &\leq \sum_{i=0}^{d-1} \Pr[(f(u) = i) \,|\, M_d] \cdot \Pr[(A_i \leq k-1) \,|\, M_i]\end{aligned}$$

which concludes the proof of the lemma. □

It remains to lower bound the number of hops taken by the GREEDY algorithm. Assume as before that the initial node is $||z|| = n$ and the destination is $\vec{0}$. Divide the nodes of the graph into sets $B_0, B_1, \ldots, B_{\log n}$ according to their distance from $\vec{0}$ (or L_1 norm), such that $B_i = \{u|2^{i-1} \leq ||u|| < 2^i\}$. So $\vec{0} \in B_0$ and $z \in B_{\lceil \log n \rceil}$. We slightly change the GREEDY algorithm thus: if the algorithm reaches a node within a ball B_i it is granted access to all nodes with distance at least 2^{i-1} from 0, i.e. to all nodes in $B_i, B_{i+1}, ..., B_n$. When routing in a skip graph the algorithm is also given the membership vectors of these nodes. The reason for this change is to cancel the dependencies on previous hops, it may only reduce the number of hops GREEDY takes, since it allows the algorithm a 'free' hop to the edge of the ball B_i. For each $0 \leq i \leq \log n$ let X_i be the indicator of the event: "The path taken by GREEDY includes at least one vertex in B_i". Clearly the number of nodes in the path is at least $\sum_{i=o}^{\log n} X_i$.

Lemma 4.5.3. *Both for skip graphs and for small world graphs and for each i, it holds that*

$$\Pr[X_i = 1|X_{i+1}, X_{i+2}, \ldots, X_{\log n}] \geq c$$

for some constant c independent of n.

Before proving the lemma we show why it suffices to prove Theorem 4.5.1. Let Y_i be a Bernoulli variable with $\Pr[Y_i = 1] = c$. Now $\mathbb{E}[\sum Y_i] = c \log n \leq \mathbb{E}[\sum X_i]$. Furthermore the random variable $\sum X_i$ dominates the random variable $\sum y_i$. We have

$$\Pr[\sum X_i \leq \frac{1}{2} c \log n] \leq \Pr[\sum Y_i \leq \frac{1}{2} c \log n] \leq \frac{1}{n^\epsilon}$$

according to Chernoff's bounds.

Proof of Lemma 4.5.3. As before we distinguish between the two cases:

Small World Percolation Networks Assume that the values of $X_{i+1}, \ldots, X_{\log n}$ are already set and that j is the smallest index such that $X_j = 1$. Since we changed the algorithm such that when a ball B_i is reached all nodes in it are revealed, it is clear that X_i is independent from $X_{j+1}, X_{j+2}, \ldots, X_{logn}$, it remains to analyze $\Pr[X_i = 1 | X_{i+1} = 0, X_{i+2} = 0, \ldots, X_j = 1]$. Let y be the node in B_j which is closest to 0, i.e. the node probed by GREEDY. The notation $y \sim B_i$ stands for the event - 'y is connected by an edge to B_i'. For convenience let $B = \cup_{j=0}^{i-1} B_j$. All edges are independent of each other. Therefore $\Pr[X_i = 1 | X_{i+1} = 0, X_{i+2} = 0, \ldots, X_j = 1]$ is the probability y is connected to B_i and is not connected to $B_0, B_1, ..., B_{i-1}$, conditioned on it being connected to one of them. We need to compute:

$$\begin{aligned}\Pr[y \sim B_i \wedge y \not\sim B | y \sim B \cup B_i] &= \frac{\Pr[y \sim B_i \wedge y \not\sim B]}{\Pr[y \sim B \cup B_i]} \\ &= \frac{\Pr[y \sim B_i] \cdot \Pr[y \not\sim B]}{1 - Pr[y \not\sim B] \Pr[y \not\sim B_i]}\end{aligned}$$

It is easy to verify that $\Pr[y \not\sim B] \geq \Pr[y \not\sim B_i]$ and that $\Pr[y \not\sim B] \geq \epsilon$ for some constant ϵ. We have:

$$\frac{\Pr[y \sim B_i] \cdot \Pr[y \not\sim B]}{1 - Pr[y \not\sim B] \Pr[y \not\sim B_i]} \geq \frac{(1 - \Pr[y \not\sim B_i]) \Pr[y \not\sim B]}{(1 - \Pr[y \not\sim B_i])(1 + \Pr[y \not\sim B])} \geq \frac{\epsilon}{1 + \epsilon}$$

Skip Graphs Here we have to deal with some dependencies. Let D denote the event that the algorithm reached the node y (i.e. the segment B_j which contains y). As before we need to compute:

$$\frac{\Pr[(y \sim B_i \wedge y \not\sim B) | D]}{\Pr[y \sim B \cup B_i | D]}$$

The events $\{y \sim B_i\}$ and $\{y \not\sim B\}$ are positively correlated, even when conditioned on D. So the calculation of the previous section applies here as well. □

4.6 Implementations of NoN

We ran simulations in which we compared the performance of the GREEDY algorithm and the performance of the NoN-GREEDY algorithm. We constructed a skip graph of up to 2^{17} nodes and a small world percolation graph of up to 2^{24} nodes. In a small world graph it is not necessary to create the full graph in advance. Each time the message reached a node, we randomly created the neighborhood of radius 2 around the node. This is a negligible compromise over the definition of the model, since the edge in which the node was entered might not be sampled. This technique allowed us to run simulations on much larger graphs. For each graph size we ran 150 executions. A substantial improvement could be seen. Figures 4.4 and 4.5 demonstrate an improvement of about 48% for skip graphs of size 2^{17} and an improvement of 34% for small world percolation graphs of size 2^{24}. Figure 4.4 also depicts the average shortest path in the graph. We see that the shortest paths may be 30% shorter than the paths found by NoN, yet even for moderate network sizes, the NoN algorithm performs substantially better than then GREEDY.

An even more impressive improvement could be seen when the size of the graph is fixed and the average degree changes. We fixed a small world percolation graph of size 2^{20}. After that we deleted each edge with a fixed probability which varied from 0 to 0.9 (a graph with roughly one tenth of

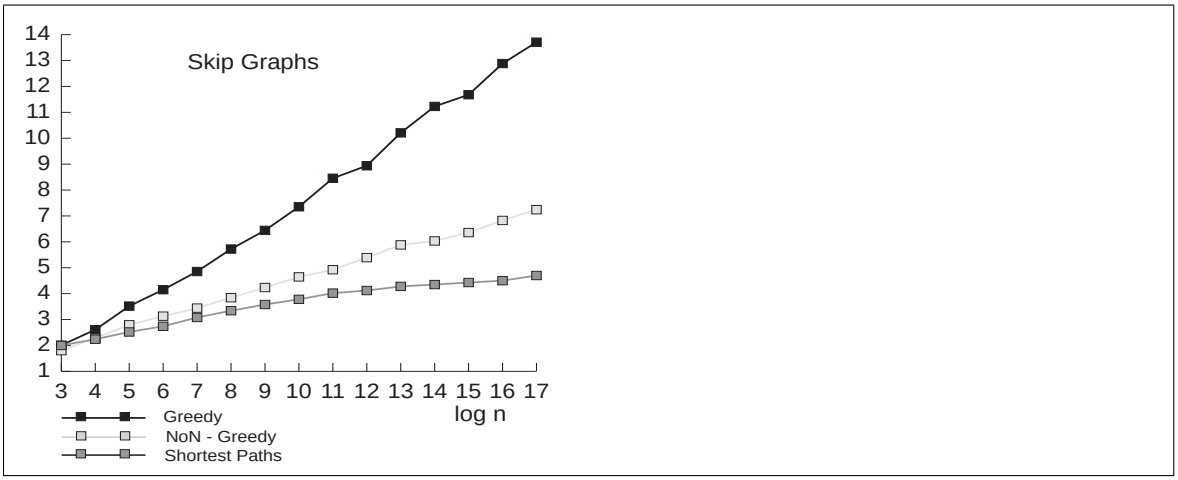


Figure 4.4: Average length of GREEDY paths, NoN paths and shortest paths in skip graphs

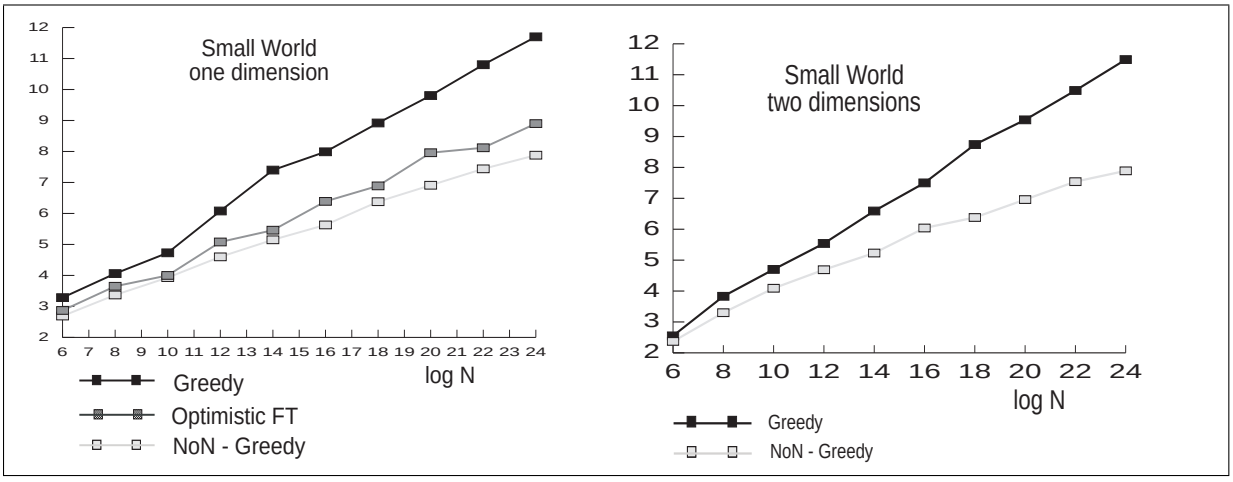


Figure 4.5: Average length of hops in Small World Percolation graphs of dimensions $1, 2$.

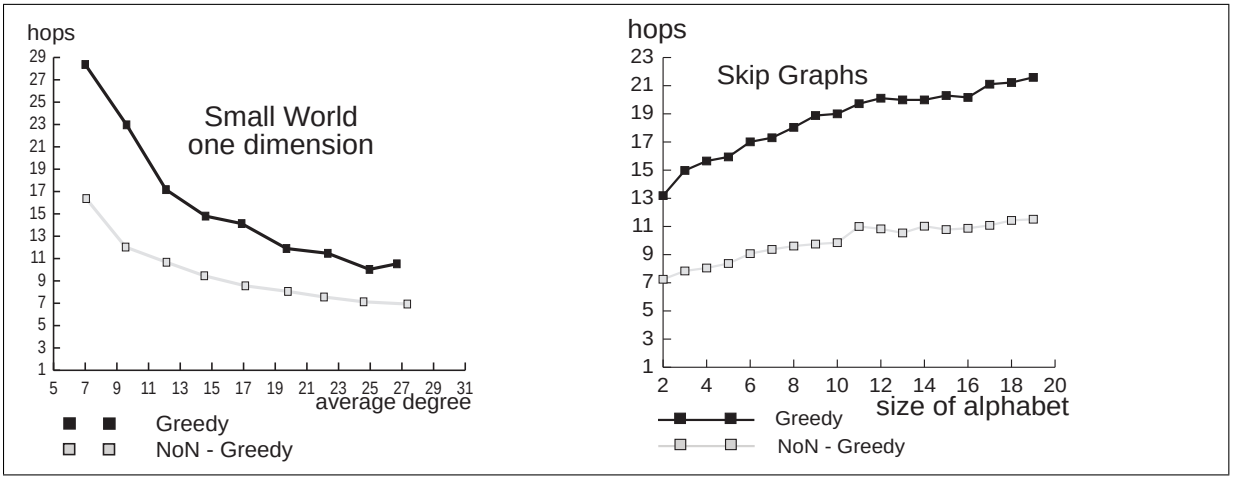


Figure 4.6: The tradeoff between average degree and path length in a Small World Percolation graph with 2^{20} nodes (left) and skip graphs of 2^{17} nodes (right).

the edges). Figure 4.6 depicts the results of these simulations. It shows that the reduction in the number of hops is more or less independent from the number of edges. The path length achieved by the GREEDY algorithm when the degree is 26 is achieved by the NoN algorithm when the degree is merely 12. In the case of skip graphs we ran the simulation for a graph of size 2^{17} and varied the size of the alphabet of the membership vectors. When the alphabet size is s the average degree is $O(\log_s n)$. We can see in Figure 4.6 that NoN with alphabet size 20 is better than Greedy with alphabet size 2, i.e. when the average degree is $\log_2 20 \simeq 4.3$ bigger.

4.6.1 A different Implementation

The algorithm presented in Figure 4.1 is somewhat unnatural. Each NoN step has two phases. In the first phase the message is sent to a neighbor whose neighbor is close to the target. In the second phase a greedy step is taken (i.e. the message moves to the neighbor of neighbor). A 1−phase implementation would let each node initiate a NoN step again, i.e. each node upon receiving a message, finds the closest neighbor of neighbor, and passes the message on. This variant is harder to analyze, indeed Theorem 4.4.2 holds for the 2−phase version only. Yet, as Figure 4.7 shows, in practice the two variants have basically the same performance.

4.6.2 System Issues with NoN-greedy

An implementation of the NoN-GREEDY algorithm in a P2P network 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 maintaining the TCP links between nodes. This cost remains unchanged, irrespective of which routing protocol we use: GREEDY or NoN-GREEDY. The primary concern in implementing NoN-GREEDY is the amount of communication-overhead needed to keep the neighbor-of-neighbor lists (reasonably) up-to-date. Updates could be piggy-backed on top of maintenance messages (the "keep-alive" messages). Moreover, the neighbor-

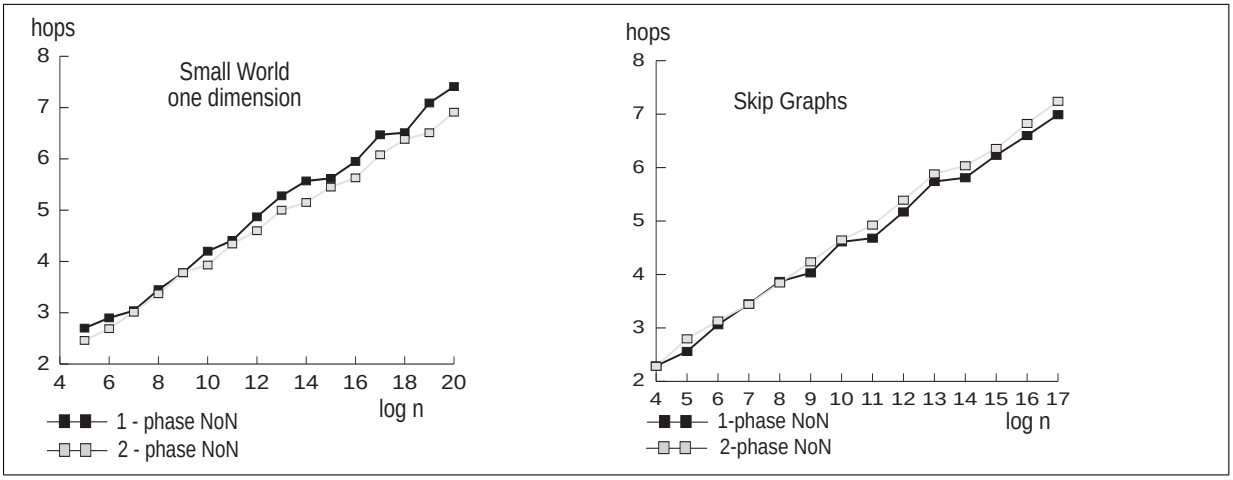


Figure 4.7: Comparison between the two variants of NoN

of-neighbor information at a node does not have to be perfectly up-to-date at all times to derive the benefits of NoN-GREEDY routing, as could be seen in the next Section.

In the following we analyze more carefully the amount of communication needed in order to keep the routing lists up-to-date. The execution NoN requires that nodes should update each other regarding their own lists of neighbors. Such an update occurs in two scenarios:

1. Each node upon entrance sends its list of neighbors to its neighbors.

2. Whenever a node encounters a change in its neighbor list (due to the entrance or exit of a node), it should update its neighbors.

The extra communication imposed by these updates is not heavy due to the two following reasons. First, assume nodes u, v are neighbors. Node u periodically checks that v is alive (for instance by pinging it). Checking whether v's neighbor list has changed could be piggy-backed on the maintenance protocol by letting v send a hash of its neighbor list. A possible hash function may be `MD5` (though the cryptographic properties of this hash function are not needed). Another possibility is simply to treat the id of neighbors as coefficients of a polynomial, and evaluate this polynomial at a random point. Either way the actual cost in communication is very small. When an actual update occurs there is no reason for v to send its entire neighbor list. It may only send the part of it which u misses. If it does not know which part it is then u, v may participate in a very fast and communication efficient protocol that reconciles the two sets, see e.g. [95] for details. The second reason the communication overhead is small is that the the actual updates are not urgent (as the next section will show) and may be done when the system is not busy.

It is important to notice that the implementation of the NoN algorithm does not affect the Join/Leave operations - the NoN updates should be passed only *after* the node enters the system. We conclude that implementing NoN has little cost both in communication complexity and in internal running time. It is almost a free tweak that improves performance considerably and may be implemented on top of existing constructions.

4.6.3 Fault Tolerance

The previous simulations assumed that the list of neighbor's neighbors each node holds is always correct. In reality this might not be the case. We examine two scenarios which capture the two extremes of this problem.

Optimistic Scenario: In this case we assume that a node knows whether its neighbors of neighbors lists are up-to-date or not. Whenever a node has a stale list it avoids a NoN hop and performs instead a greedy step based on its own neighbors list. We ran simulations in which each node performs with probability $\frac{1}{2}$ a NoN step, and with probability $\frac{1}{2}$ a greedy step. Whenever a NoN step is performed, both phases of it are performed correctly. Figures 4.5 and 4.8 show that the total performance is hardly compromised. A small world of size 2^{22} suffered a relative delay of less than one hop, A skip graph of size 2^{17} suffers a relative delay of 1.2 hops. But why is the optimistic scenario justified? Our suggestion is that each node would calculate a hash of its neighbors list. This hash would be sent to all its neighbors on top of the maintenance messages. Thus with a minuscule overhead in communication each node would know whether its lists are up-to-date.

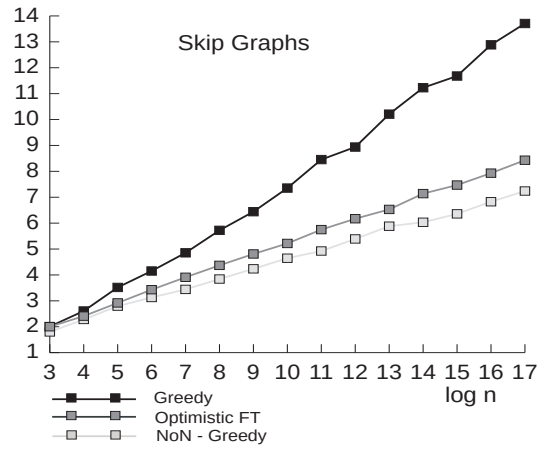


Figure 4.8: Optimistic fault tolerance in Skip Graphs

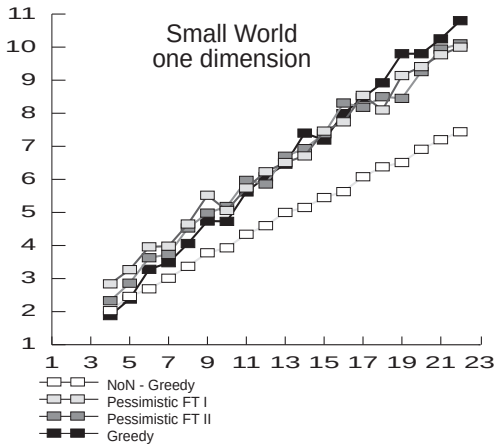


Figure 4.9: Pessimistic Fault Tolerance in Percolation Small World graphs

Pessimistic Scenario: In this scenario we assume that a node is unaware that its neighbor's neighbors lists are not up-to-date. So when node u passes a message to node w expecting it to move on to node z, with probability $\frac{1}{2}$ the edge (w, z) no longer exists. We tested two variants: in the first one, whenever this occurs the intermediate node w performs a greedy step. In the second variant the intermediate node w initiates another NoN step. The results of the simulations appear in Figure 4.9. It could be seen that in the pessimistic scenario, the performance of NoN is approximately the same as the Greedy algorithm.

We conclude that the NoN-GREEDY algorithm is beneficial even if the neighbor of neighbor lists are error prone.

4.7 Discussion

Randomization Reduces Latency: A common strategy in the design of P2P routing networks is to first identify a static graph which is known to possess good properties, and then to adapt the static graph topology to handle the dynamism (arrival/departure of nodes) and scale (changes in the average number of nodes). The resulting dynamic routing network resembles the underlying static graph closely. In the case of skip graphs, a 'perfect' skip graph has the i^{th} edge of each node cover a distance of 2^i, i.e., the lengths of edges of a node form a geometric series. The randomization involved in the dynamic construction is usually considered as a negative by-product and much effort is put in reducing it. For instance, a deterministic P2P routing network that guarantees that the skip graph is almost 'perfect' is presented in [56]. As was noticed by Harvey *et al* [57], a perfect skip graph is similar to Chord [120]. The average length of shortest paths in both Chord (studied in [45]) and hypercubes is $\Omega(\log n)$. Xu *et al* [125] proves that if edges are added to the cycle deterministically such that the existence of an edge (x, y) is a function of $|x - y|$ (and not say x, y themselves), then the diameter of a network of degree $\log n$ is bounded by $\Omega(\log n)$. This leads to the following counter-intuitive and surprising fact:

> *Randomization of edges* reduces *the average length of shortest paths in the hypercubes, Chord and Skip Graphs.*

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 NoN-Greedy algorithm 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.

Chapter 5

The Expansion and Mixing Time of Skip Graphs

Summary: We prove that with high probability a skip graph contains a 4-regular expander as a subgraph and estimate the quality of the expansion via simulations. As a consequence skip graphs contain a large connected component even after an adversarial deletion of nodes. We show how the expansion property could be used to sample a node in the skip graph in a highly efficient manner. We also show that the expansion property could be used to load balance the skip graph quickly. Finally it is shown that the skip graph could serve as an unstructured P2P system, thus it is a good candidate for a hybrid P2P system.

5.1 Introduction

Skip graphs [16] or SkipNets [57] are randomized distributed data structures designed for use in peer-to-peer (P2P) storage systems. Like Distributed Hash Tables (DHTs), skip graphs scale gracefully, and as seen in previous chapter, offer excellent query complexity. Skip graphs have an advantage over DHTs in the sense that they directly support *range queries*, while DHTs provide exact search only. For a more thorough description of skip graphs we refer the reader to Section 4.4.1. Much of the usefulness of skip graphs depends on their properties as random graphs. It was previously shown [16] that (with high probability) skip graphs have expansion ratio $\Omega(1/\log n)$: every subset of $m \leq n/2$ nodes of a skip graph has $\Omega(m/\log n)$ neighbors. This bound is surprisingly low given that skip graphs have average degree $O(\log n)$, but experimental examination of small cases suggested it was the best possible.

In this chapter we prove that with high probability a skip graph has a node expansion ratio of $\Omega(1)$: every subset of $m \leq n/2$ nodes has $\Omega(m)$ neighbors. In fact, we prove a stronger result: with high probability, a skip graph contains a spanning degree-4 regular expander as a subgraph; i.e., it contains a spanning degree-4 regular subgraph with expansion ratio $\Omega(1)$. Each node can compute which 4 of its edges belong to the expander using only local information in $O(1)$ expected time and $O(\log n)$ time with high probability. Consequences of the embedded expander (which are analyzed in Sections 5.3 and 5.4) include:

- **Fault tolerance:** The expansion property is equivalent to the property that a deletion of k nodes may isolate from the primary component at most $O(k)$ nodes. In other words, even if a constant fraction of the nodes are deleted by an *adversary*, still a constant fraction of the

nodes would remain connected in a single component.

- **Efficient sampling:** A random walk in an expander graph quickly converges towards the stationary distribution. This could be used in order to sample uniformly a random node. We present several sampling algorithms and show that they are much faster than the currently best known algorithms.

- **Low hitting times:** Random walks on expanders have the property of hitting a large set of nodes fast and with high probability. This can be used for a variety of applications such as load balancing, gathering statistics on the nodes of the skip graph, and for finding highly replicated data items. It is known that unstructured P2P systems which are expanders permit more efficient searches than simple flooding [48]. The skip graph therefore is shown to be competitive with *unstructured* P2P systems, thus making it an excellent candidate for a hybrid P2P system.

5.1.1 Comparison with previous work

Expanding networks The advantages of an expanding topology are well known, and the literature is abundant with variations of expanding networks. In the context of dynamic P2P networks we are aware of only two previous approaches. The first was presented in Section 2.7.3, we used the continuous-discrete approach to build overlay network that emulates the Margulis [89, 44] explicit construction of expanders. The quality of the expansion property depends upon the smoothness of the i.d. selection scheme. The expanding network does not currently support an efficient lookup functionality though a step towards achieving this was given by Larsen [72]. The main advantage of this construction is its guaranteed expansion. Its main drawback compared to the present work is that it has a rather large overhead in maintaining the network and keeping the i.d. selection well balanced, thus making it an appealing theoretical solution yet somewhat unpractical.

The second suggestion for an expanding overlay network was made by Law and Siu [73]. They suggest building d random Hamiltonian cycles, which have an optimal spectral gap w.h.p. and are thus expanding [43]. The main advantages of this scheme are its relative simplicity and its optimality in the sense that w.h.p. the graph will have the (almost) largest possible spectral gap with respect to the degree. The construction does not support a lookup operation.[1] Their construction also needs a sampling protocol as a primitive, and assumes that samples are obtained by performing a random walk. But the uniformity of the distribution produced by the random walk depends on the expansion, while the expansion depends on the uniformity. This mutual dependency means that an error in the early stages of the construction may accumulate and ruin the expanding property. In contrast, the expansion in our construction depends upon the random bits generated independently by each node separately, so there is no mutual dependency between the correctness of the join algorithm and the expansion. Furthermore, it should be noted that Law and Siu's construction can easily be implemented on top of the skip graph using the sampling algorithm of Section 5.3, obtaining the best properties of both constructions. Indeed this is implicitly done by Zatloukal and Harvey [126] which build a variant of skip graphs with two random Hamiltonian cycles.

[1] In order to support lookup their construction would need logarithmic degree and then a lookup takes $\Theta(\log n)$ hops on expectation.

Sampling schemes In the context of P2P systems, a random walk sampling scheme was previously suggested by Law and Siu [73] (as mentioned above) and by Gkantsidis [48]. Obtaining good samples using such random walks requires *a priori* knowledge of the spectral gap of the graph. In Section 5.3.2, we show that the spectral gap of the skip graph is well concentrated and therefore can be known in advance, so that random walks work well in a skip graph.

A different sampling scheme for DHTs was suggested by King and Saia [68]. Their scheme yields an *exact* uniform distribution and runs in expected logarithmic time. Recent work by King, Lewis and Saia [67] shows that the expected running time is at least $11 \log n$. Empirical testing shows that our algorithm runs much faster, albeit at the cost of slight deviations from uniformity. A running time of about $2 \log n$ produces a sample from a distribution that is close enough to uniform for most conceivable applications (see Section 5.3.2).

5.2 Main result

We will show that skip graphs have an edge expansion of some small $\epsilon > 0$ with high probability. Throughout the paper let G denote a skip graph of n vertices. For a vertex set U define $\delta(U)$ to be the number of nodes outside U which have a neighbor in U.

Theorem 5.2.1. *There exists an $\epsilon > 0$ independent of n such that with high probability[2] the following event occurs: G has a subgraph G' of degree 4 such that for every set $U \subset V$ of size at most $\frac{n}{2}$ it holds that $|\delta(U)| \geq \epsilon |U|$.*

Remark: We do not state what is the largest ϵ for which Theorem 5.2.1 is correct. Indeed, in the proof we are liberal in making ϵ as small as necessary. A lower bound on the expansion is obtained via simulations. See Section 5.3.2.

The subgraph G' in the union of two degree 2 spanning sub graphs. In a skip graph, all the nodes are connected by the bottom-layer cycle S_ϵ. The graph G' is the union of S_ϵ with another sub graph. The second sub graph is a collection of cycles which we call *buckets*, that are obtained by selectively including higher-level cycles as described in Section 5.2.2. An important property of the buckets is that the event that they expand a set is *independent* from the event that S_ϵ expands a set, so the effect of each sub graph can be analyzed separately. The idea of the proof is to show that the probability a set $U \subset V$ does not expand by the cycle S_ϵ *and* by the buckets is sufficiently small.

5.2.1 The expansion of the cycle S_ϵ

We show that for most sets the cycle edges alone suffice to show expansion. The remaining sets expand due to the buckets. For a set of vertices A denote by $c(A)$ the set of cycle edges which have exactly one end point in A. The following Lemma is proven in [16]:

Lemma 5.2.2. *In a n-node skip graph, the number of sets A where $|A| = m < n/2$ and $|c(A)| \leq s$ is at most*

$$2\binom{m+1}{s}\binom{n-m-1}{s}$$

[2] Throughout the paper the term 'with high probability' stands for probability $1 - n^{-\delta}$ for some $\delta > 0$

Take s to be ϵm for a sufficiently small ϵ. we have:

$$2\binom{m+1}{\epsilon m}\binom{n-m-1}{\epsilon m} \leq \exp\left(0.1m\ln\frac{n}{m}\right) \tag{5.1}$$

Since S_ϵ is had degree 2, for every set A, the number of *nodes* expanded by S_ϵ is at least $\frac{1}{2}|c(A)|$. Conclude that far small enough ϵ, the number of sets A for which $|\delta(A)| \leq \epsilon|A|$ is at most $\exp\left(0.1m\ln\frac{n}{m}\right)$.

5.2.2 The Buckets

The second sub graph is obtained by partitioning the nodes among a disjoint family of *buckets*, upper-level cycles S_σ that are not too small.[3] A cycle S_σ is a bucket if σ is a minimal prefix for which $|S_\sigma| \geq 10$ and either $|S_{\sigma 0}| < 10$ or $|S_{\sigma 1}| < 10$ (or both). Equivalently, the buckets are the cycles obtained by repeatedly splitting cycles, starting with the original cycle S_ϵ, by adding one bit at a time to the common prefix, stopping only when further divisions would yield cycles that are too small. The minimum bucket size is set to 10 for convenience in the proof, other values may be chosen as well. In simulations, we show that a bucket size of 4 appears to be the best choice.

We call the edges that create the cycles of each bucket the *bucket edges* of the graph. The following observation motivates the division into buckets: Consider a set A of nodes. Whenever there exists a bucket which contains a node in A and a node not in A, the bucket contributes at lease one edge to $\delta(A)$. Our aim therefore is to prove that with high probability there are at least $\epsilon|A|$ buckets that are partially covered by A; i.e. that A hits at least one element and misses at least one element from each bucket.

Lemma 5.2.3. *With high probability for all* $1 \leq m \leq \frac{n}{2}$ *the number of nodes in buckets which contain more than* $100n^{\frac{1}{4}}m^{-\frac{1}{4}}$ *nodes is at most* $\frac{m}{10}$.

Before proving Lemma 5.2.3, we show how to deduce Theorem 5.2.1 given that the event described in the lemma occurs. Let ϵ be a small constant. We calculate for how many sets of size m the buckets do not contribute ϵm edges to the expansion.

Call buckets which contain more than $100n^{\frac{1}{4}}m^{-\frac{1}{4}}$ nodes *large buckets*. The rest of the buckets will be referred to as *small*. We do not count edges caused by large buckets (thus overcounting the number of bad sets). According to Lemma 5.2.3, there are at most $\frac{m}{10}$ nodes in large buckets, therefore there are at most $2^{m/10}$ ways to place the nodes in the large buckets. Since each bucket contains at least 10 nodes, there at most $n/10$ buckets. There are at most $\binom{n/10}{\epsilon m}$ ways to choose the ϵm small buckets that do expand. Each small bucket is of size at most $100n^{\frac{1}{4}}m^{-\frac{1}{4}}$. It follows that there are at most $\binom{\epsilon 100n^{\frac{1}{4}}m^{\frac{3}{4}}}{m}$ ways to place the vertices in these ϵm small buckets. The remaining vertices are scattered in other buckets, with the restriction that each such bucket is either not hit by the set or is covered completely by it. Each bucket is of size at least 10, therefore there are at most $\binom{n/10}{m/10}$ ways to choose the location of the rest of the vertices. We conclude that the total number of bad sets is bounded by:

[3] Recall that S_σ is the doubly-linked list of all nodes whose membership vectors have σ as a prefix.

$$
\begin{aligned}
&2^{m/10}\binom{n/10}{\epsilon m}\binom{100\epsilon n^{1/4}m^{3/4}}{m}\binom{n/10}{m/10} \\
&\leq \exp\left((m/10)\ln 2 + \epsilon m \ln\left(\frac{en}{10\epsilon m}\right) + m\ln\left(100e\epsilon n^{1/4}m^{-1/4}\right) + (m/10)\ln\left(\frac{en}{m}\right)\right) \\
&\leq \exp\left(m\left(\epsilon\ln\left(\frac{n}{m}\right) + \frac{1}{4}\ln\left(\frac{n}{m}\right) + \frac{1}{10}\ln\left(\frac{n}{m}\right) + 0.2\right)\right) \quad \text{for small enough } \epsilon \\
&\leq \exp\left(0.8m\ln\frac{n}{m}\right). \qquad (5.2)
\end{aligned}
$$

In the first inequality we use the fact that $\binom{n}{m} \leq \left(\frac{ne}{m}\right)^m$. The total number of sets of size m is $\binom{n}{m} \geq \left(\frac{n}{m}\right)^m$. Inequality 5.1 states that the number of sets which are not expanded by the cycle edges is at most $\binom{n}{m}^{0.1}$. Inequality 5.2 states that the probability a set of size m is not expanded by the bucket edges is at most $\binom{n}{m}^{-0.2}$. The key observation is that the large cycle is *independent* from the division into buckets. In other words, all bad sets with respect to the large cycle are equally likely to be expanded by the bucket edges. We conclude that the probability there exists a set of size m which is not expanding is at most $\binom{n}{m}^{-0.1}$. The proof of Theorem 5.2.1 is completed by union bounding these probabilities for all $2 \leq m \leq \frac{n}{2}$ and the error probability of Lemma 5.2.3.

We now proceed with the proof of Lemma 5.2.3:

Proof. of Lemma 5.2.3

Let M denote all the prefixes of length $\lfloor \log n - \log\log n - 3 \rfloor$, so that $\frac{n}{16\log n} \leq |M| \leq \frac{n}{8\log n}$. For every $\sigma \in M$, let B_σ denote all the nodes that have σ as a prefix.

Lemma 5.2.4. *For every $\sigma \in M$, with high probability $\log n \leq |B_\sigma| \leq 24\log n$.*

Proof. This is a simple balls and bins argument. For each $\sigma \in M$, $|B_\sigma|$ has the Binomial distribution and $8\log n \leq \mathbb{E}[|B_\sigma|] \leq 16\log n$. By Chernoff's bound:

$$
\begin{aligned}
\Pr[|B_\sigma| \leq \log n] &\leq \exp\left(-\frac{\mathbb{E}[|B_\sigma|]}{4}\right) \leq \frac{1}{n^2} \\
\Pr[|B_\sigma| \geq 24\log n] &\leq \exp\left(-\frac{\mathbb{E}[|B_\sigma|]}{8}\right) \leq \frac{1}{n}
\end{aligned}
$$

□

We conclude that w.h.p. all buckets are of size at most $24\log n$, therefore the lemma is correct if $m \leq \frac{n}{\log^4 n}$. Assume to the contrary that $m > \frac{n}{\log^4 n}$. Suppose further that during the procedure of creating the buckets we have a bucket of size ℓ which corresponds to the prefix σ. The bucket does not split into two buckets if there are less than 10 nodes with prefix $\sigma.0$ or less than 10 nodes with prefix $\sigma.1$. Conclude that the probability a bucket of size ℓ is not partitioned into two buckets of size at least 10 is

$$
2\left(\binom{\ell}{0} + \binom{\ell}{1} + \ldots + \binom{\ell}{10}\right)2^{-\ell} \leq 5\ell^{10}2^{-\ell}.
$$

Let p_k denote the probability an element belongs to a bucket of size at least k. Once a node participates in more than $n-k$ partitions, it must belong to a bucket of size smaller than k, so we

have:

$$p_k \leq \sum_{\ell=k}^{n} 5\ell^{10}2^{-\ell} \leq 20k^{10}2^{-k} \tag{5.3}$$

According to Lemma 5.2.4, we can divide the nodes into sets according to the first $\lfloor \log n - \log\log n - 3 \rfloor$ bits of their prefix. Each set is of size at most $24\log n$, and there are at most $\frac{n}{8\log n}$ such sets. Denote by Z_i the random variable counting the number of nodes in the ith set which will eventually be in a bucket of size at least k. We know the following:

1. For each i it holds that $0 \leq Z_i \leq 24\log n$.
2. All the Z_i are mutually independent.
3. Inequality (5.3) implies that $\mathbb{E}[\sum Z_i] = np_k \leq 20nk^{10}2^{-k}$.

We use the following version of the Chernoff/Hoeffding bound:

Theorem 5.2.5. *For mutually independent random variables $Z_1, \ldots, Z_\ell$, where $Z_i \in [a, b]$*

$$\Pr\left[\left|\sum_{i=1}^{\ell} Z_i - \mathbb{E}\left[\sum_{i=1}^{\ell} Z_i\right]\right| \geq \ell\delta\right] \leq 2e^{-\frac{2\delta^2}{(b-a)^2}\cdot\ell}$$

Set $k = 100\left(\frac{n}{m}\right)^{\frac{1}{4}}$ and $\mu = \mathbb{E}[\sum Z_i] \leq 20nk^{10}2^{-k}$ and we have:

$$\Pr\left[\sum Z_i \geq \frac{m}{10}\right] \leq \Pr\left[\left|\sum Z_i - \mu\right| \geq \mu - \frac{m}{10}\right]$$

We can use Theorem 5.2.5 by setting $(b-a) = 24\log n$ and $\ell = \frac{n}{24\log n}$ and $\delta = (\frac{m}{10} - \mu)\frac{\log n}{n}$. We have:

$$\begin{aligned} &\leq 2\exp\left(-\frac{1}{24^2\log^2 n}\cdot 2\left(\frac{m}{10}-\mu\right)^2\frac{\log^2 n}{n^2}\cdot\frac{n}{24\log n}\right) \\ &\leq 2\exp\left(-\left(\frac{m}{10}-\mu\right)^2\cdot\frac{1}{24^3 n\log n}\right) \end{aligned} \tag{5.4}$$

Next we bound μ as a fraction of m. Substitute $k = 100\left(\frac{n}{m}\right)^{\frac{1}{4}}$ and $\mu \leq 20nk^{10}2^{-k}$ and we have::

$$\begin{aligned} \frac{m}{10} - \mu &\geq \frac{m}{10} - 2000n\left(\frac{n}{m}\right)^{\frac{10}{4}}\cdot 2^{-100\left(\frac{n}{m}\right)^{1/4}} \\ &\geq m\left(\frac{1}{10} - 2000\cdot 2^4\cdot 2^{-100}\right) \geq 0.09m \end{aligned}$$

Now we complete the calculation of Equation (5.4) using the assumption that $m \geq \frac{n}{\log^4 n}$.

$$\leq 2\exp\left(-(0.09m)^2\cdot\frac{1}{24^3 n\log n}\right) \leq 2\exp\left(-\frac{n}{20\cdot 10^5\ln^5 n}\right)$$

The proof of Lemma 5.2.3 is completed by union bounding for all m. □

5.2.3 Identifying the bucket edges:

Each node can identify its bucket edges in $O(1)$ time in expectation and $O(\log n)$ time with high probability via the following procedure: initially each node sets its maximal edge as a bucket edge; i.e., assumes that the prefix of its bucket is the longest prefix for which it has an edge. Next each node performs a walk along the bucket's cycle to verify that the bucket's size is large enough i.e. that the bucket contains at least 10 nodes[4]. While performing this check, the node updates the other nodes along the cycle about the bucket edge. Now there are a few cases:

1. If the bucket is of size less than 10 then the prefix of the bucket is too long. So the node picks the next longest edge as its bucket edge and again counts the size of the bucket's cycle.

2. It may be that even though the bucket's size is more than 10 nodes, a node is informed that its current bucket edge is not valid and that it has to reduce the length of the bucket edge. This case occurs if the bucket which corresponds to the same prefix with the last bit converted is too small; i.e. a different node performed case number (1).

3. If the bucket size is at least 10 and case (2) does not occur then the bucket edges are decided upon.

The running time of the algorithm is in the order of the size of the bucket, therefore the algorithm runs in expected constant time, and in logarithmic time with high probability.

5.3 How to sample a random node

A random walk on a regular expander mixes rapidly, and may be used to sample a node in the skip graph efficiently and simply. In the following it is convenient to think of distributions over the nodes as vectors. We say that $\vec{p}$ is a *distribution vector* if $p_i \geq 0$ for all $0 \leq i < n$, and $\sum p_i = 1$. Let $\vec{u}$ denote the probability vector of the uniform distribution over the nodes; i.e. $\vec{u} = (\frac{1}{n}, \ldots, \frac{1}{n})$. Let $\vec{p}$ be some arbitrary distribution over the nodes. A useful measure for the distance between two distribution is the *variation distance*[5] which is defined to be half of the L_1 distance between their vectors; i.e. $\Delta(\vec{p}, \vec{u}) = \frac{1}{2}||\vec{p} - \vec{u}||_1 = \frac{1}{2}\sum_v |p_v - \frac{1}{n}|$. Assume a random walk is performed on a $d-$regular graph, starting from some arbitrary initial distribution $\vec{p}$ (distribution $\vec{p}$ may of course put all its weight on a single vertex). Let A be the adjacency matrix of the $d-$regular graph and let $\hat{A} = \frac{1}{d}A$. The largest eigenvalue of A is d (and $\vec{u}$ is an eigenvector), so the largest eigenvalue of $\hat{A}$ is 1. Denote by α the second largest eigenvalue of $\hat{A}$. Now for every integer t the vector $\hat{A}^t\vec{p}$ is the distribution over the nodes after performing t steps of a random walk. It holds (see [11]) that if the graph has a node or edge expansion which is bounded away from 0 then α will be bounded away from one.[6] The following theorem is well known (see for instance [10]):

Theorem 5.3.1. *For every initial distribution $\vec{p}$, it holds that*

1. $||\hat{A}^t\vec{p} - \vec{u}||_1 \leq \sqrt{n}\alpha^t \cdot ||\vec{u} - \vec{p}||_2$ *where $||\cdot||_i$ stands for the L_i norm.*

[4] If we allow buckets of size 2 then this part is not necessary

[5] There are many ways to define distance between distributions. Variation distance is the most useful in our context.

[6] We slightly abuse notation. The statement is meaningful only when discussing families of graphs with $n \to \infty$ (which is of course our case), and not a single graph.

2. *If* $t > \frac{\log(1/\delta)}{\log(1/\alpha)}$ *then for every node* v *it holds that* $|(\hat{A}^t\vec{p})_v - \frac{1}{n}| \leq \delta$. *In particular, if* $\delta = \frac{1}{2n}$ *then the probability each node is sampled by the walk is in the range* $[\frac{1}{2n}, \frac{3}{2n}]$.

Theorem 5.3.1 combined with Theorem 5.2.1 implies that a long enough random walk along the subgraph formed by the cycle edges and the bucket edges yields an approximately uniform sample. The length of the random walk may depend upon the application. If it is required that each node be sampled with probability which is within factor 2 of uniform then, by the second assertion of the theorem, a walk of length $\frac{1+\log n}{\log(1/\alpha)}$ suffices. For a variation distance between the sample and the uniform distribution bounded by ϵ, according the the first assertion a walk of length $t = \frac{\log n + 2\log(1/\epsilon)}{2\log(1/\alpha)}$ is enough (note that if $\vec{p}$ puts all its weight on one node then $||\vec{p} - \vec{u}||_2 \approx 1$).

The running time to obtain a fixed variation distance depends upon the second eigenvalue and upon $\log n$. Simulations show (see Section 5.3.2) that the second eigenvalue is concentrated around 0.85. An estimation of $\log n$ could be easily derived through simple procedures as was shown in Section 2.3.

5.3.1 Speeding up the mixing time

Theorem 5.2.1 refers to a constant degree subgraph. One might hope that the logarithmic degree of the skip graph implies that using more edges would significantly decrease the mixing time. Unfortunately this is not the case.

Lemma 5.3.2. *With high probability there exists a set* $A \subset V$ *such that* $\frac{n}{2} - \log n\sqrt{n} \leq |A| \leq \frac{n}{2}$, *and* $|\delta(A)| \leq |A|$.

Proof. For $\tau \in \{0, 1\}$ Let A_τ be the set of vertices that have τ as the first bit of their membership vector. Assume w.l.o.g that $|A_0| \leq |A_1|$. We have that w.h.p. $|A_0| \geq \frac{n}{2} - \log\sqrt{n}$ and that $|\delta(A_0)| \leq |A_0|$. □

Lemma 5.3.2 implies that the edge expansion of the entire skip graph is $O(1)$, so for every subgraph of the skip graph, the mixing time is bounded by $\Omega(\log n)$. Furthermore since the conductance of the set A_0 is $O(\frac{1}{\log n})$ the mixing time of the entire skip graph is $\Omega(\ln n \ln\ln n)$. It may be the case however that adding a small set of edges to the subgraph would improve the mixing time by a constant.

Sampling in $O(\log n/\log\log n)$ time:

The unique properties of the skip graph, in particular its support for the lookup operation, can be used to hot-start a random walk, and thus reduce the complexity of sampling. Let $\vec{u}$ denote the uniform probability. In the following we show a procedure that for a given $\delta > 0$ samples a node with distribution $\vec{p}$ such that $||u - p||_1 \leq \delta$. The expected running time is $O(\log n/\log\log n)$ where the constant depends upon δ. The procedure is as follows:

1. Choose a vector m of $3\log n$ random bits. look up the node[7] with the longest prefix which agrees with m. Call that node v.

2. Perform a random walk according to the bucket scheme, starting at v and of length $O(\log n/\log\log n)$.

[7] In case of several such nodes, any one of them suffices.

Let $\vec{p}$ be the distribution formed by the first phase of the algorithm. Our goal is to bound $||\hat{A}^t\vec{p}-u||_1$ when $t = O(\log n/\log\log n)$. Let $\vec{\epsilon} = \vec{u} - \vec{p}$. Theorem 5.3.1 states that $||\hat{A}^t\vec{p} - u||_1 \leq \sqrt{n}\alpha^t||\vec{\epsilon}||_2$. We calculate a bound on $||\vec{\epsilon}||_2$. First note that with high probability it holds that $p_v \leq \frac{2\log n}{n}$ for every node v. This is true since w.h.p. every vector of length $\log(n/2\log n)$ is the prefix of at least one node. We conclude that:

$$||\vec{\epsilon}||_2 \leq \sqrt{n \cdot \left(\frac{2\log n}{n}\right)^2} \leq \frac{2\log n}{\sqrt{n}}$$

and that $||\hat{A}^t\vec{p}-u||_1 \leq 2\alpha^t \log n$ so in order for the distance to be at most δ we need $t \geq \frac{\log\left(\frac{2\log n}{\delta}\right)}{\log(1/\alpha)}$. For example, if we take $\delta = 1/100$ and $\alpha = 0.85$ we get that $t \geq 33 + 4.3\log\log n$. As before, it is important to note that these are upper-bounds only. It may be that the walk mixes much faster. Indeed in Section 5.3.2 we show that once a random prefix is reached, a very short walk yields an almost uniform sample.

The complexity of Step (1) depends upon the algorithm used for searching a prefix. It takes $O(\log n)$ steps w.h.p. to find a prefix when the node is searched greedily (this is basically identical to the algorithm for finding the edges when joining).

5.3.2 Empirical evidence

We ran simulations in which the bucket edges and cycle edges were constructed. We calculated the second eigenvalue of the Laplacian (which is the gap between the first and second eigenvalue of the adjacency matrix). In Figure 5.1 we sketch the spectral gap as a function of $\log n$. Each entry is the average of 5 simulation only. The first observation is that the value of the second eigenvalue is extremely concentrated (which is why the average of 5 simulations is enough). When $n = 256$ the difference between the smallest gap measured and the largest was under 0.1 for all different bucket sizes. When $n = 65536$ the difference between the smallest and largest measurement was less than 0.03. Simulations show that the largest gap is achieved when the minimum bucket size is 4. In this case the spectral gap is roughly 0.495. The value of α from Theorem 5.3.1 could be calculated as: $\frac{4-0.495}{4} \approx 0.87$. The second eigenvalue of a Ramanujan graph, that is, a graph with the largest possible spectral gap is bounded by $2\sqrt{3} \approx 3.46$. In other words the largest spectral gap we could have hoped for is $4 - 3.46 = 0.54$. In this case the value α is $\frac{3.46}{4} = 0.866$, so the value of α we achieve is about 0.004 from optimal.

Next we checked the quality of the mixing by calculating the distribution over the nodes when starting from some arbitrary vertex. The vector $\hat{A}^t\vec{p}$ was explicitly calculated when $\vec{p}$ puts weight 1 on the first vertex. The simulations have $n = 2^{18}$ and bucket size of at least 4. Table 5.1 summarizes the results. The *minimum weight* column indicates the probability weight of the node least likely to be sampled, as a fraction of $\frac{1}{n}$. The *maximum weight* is the analog for the heaviest vertex. When the walk is of length $2.5\log n$ the variation distance from uniform is only 0.018 and all vertices are sampled with probability at least $0.9\frac{1}{n}$. When the walk is of length $3.5\log n$ then all vertices are sampled with probability at most $1.33\frac{1}{n}$.

It is important to note that even though the bucket edges achieve an almost optimal spectral bound, in practice they may not necessarily be the optimal choice as far as mixing is considered. For instance it may be that a random walk which uses several bucket sizes together would mix faster. When designing such heuristics one must bare in mind that the mixing time is *not* monotone in the number of edges, indeed the entire skip graph mixes slower than the expanding subgraph.

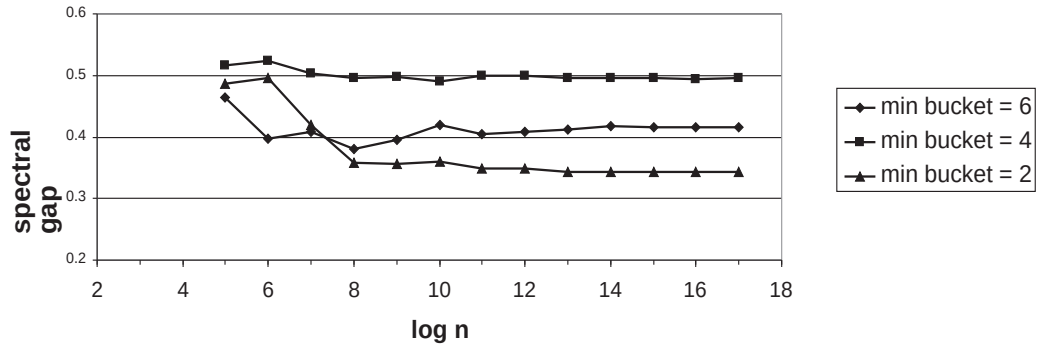


Figure 5.1: The spectral gap of the Laplacian as a function of $\log n$, using various values for the minimum bucket size.

Walk Length	*minimum weight*	*maximum weight*	*variation distance*
$1 \log n$	0.1	990	0.49
$1.5 \log n$	0.42	162	0.185
$2 \log n$	0.74	35	0.059
$2.5 \log n$	0.9	7.7	0.018
$3 \log n$	0.96	3.21	0.0055
$3.5 \log n$	0.99	1.33	0.0015
$4 \log n$	0.996	1.12	0.0005

Table 5.1: Quality of mixing when $n = 2^{18}$ and buckets are of size at least 4.

Walk Length	*minimum weight*	*maximum weight*	*variation distance*
$2 \approx 0.1 \log n$	0.05	4.95	0.14
$4 \approx 0.2 \log n$	0.41	2.75	0.08
$7 \approx 0.4 \log n$	0.67	1.72	0.04
$11 \approx 0.6 \log n$	0.83	1.29	0.017
$14 \approx 0.8 \log n$	0.9	1.16	0.01
$18 = \log n$	0.95	1.08	0.005

Table 5.2: Quality of hot-started mixing when $n = 2^{18}$, buckets are of size at least 4 and the walk starts from a random prefix.

Random walk with a hot start: We simulated a random walk starting from a *random prefix*. Table 5.2 summarizes the results. It could be seen that once a random prefix is reached, a random walk of length $7 = \lfloor 0.4 \log n \rfloor$ samples each node with probability at most $1.71\frac{1}{n}$ and at least $0.67\frac{1}{n}$. Thus, this is by far the fastest known sampling algorithm.

Estimating the expansion The simulations above may be used to give lower bound on the expansion of the graph. The following theorem is proven in [12]:

Theorem 5.3.3. *If λ is the second largest eigenvalue of a $d-$regular graph G with n vertices, then the node expansion of G is at least $\frac{2(d-\lambda)}{3d-2\lambda}$.*

Plugging in $d = 4$ and $\lambda = 4 - 0.495 = 3.505$ we get that the node expansion of the expanding subgraph is at least 0.18. The bound in Theorem 5.3.3 is probably not tight in our case, furthermore, since the expansion is monotone in the number of edges, we expect the expansion of the skip graph to be a larger constant.

5.3.3 The maximal edge heuristic

Another possible subgraph to consider is the one composed of the cycle edges and *maximal edges*. An edge (u, v) is said to be *maximal* if for either u or v it corresponds to the longest prefix that yields a nonempty cycle. Given Theorem 5.2.1, it is natural to conjecture that a random walk on these edges would mix rapidly. The main advantage of this scheme is that the maximum edge of each node is immediately identifiable without any overhead. A disadvantage is that the degrees of nodes in the resulting subgraph are not uniform, which produces a nonuniform stationary distribution.

However, this nonuniform distribution can be corrected by applying rejection sampling to the more frequently sampled high-degree nodes. We give empirical evidence that this heuristic converges quickly, achieving a distribution within 1% of uniform in just $5 \lg n$ steps.

Figure 5.2 plots the degree distribution of three different graph sizes. Each plot is the average of five simulations. The results of the simulations were very well concentrated. About 97.5% of the nodes have degrees 3 or 4. Degree 5 nodes are about 2.5% of the nodes. The remaining degrees could be found in less than 0.1% of the nodes, and in no simulation have we encountered a node with degree larger than 7. The average degree in all 15 simulations was between 3.275 and 3.285. Nodes with higher degree have a higher probability of being sampled by the walk. As mentioned previously, this can be fixed using rejection sampling based on the node degrees: when a random

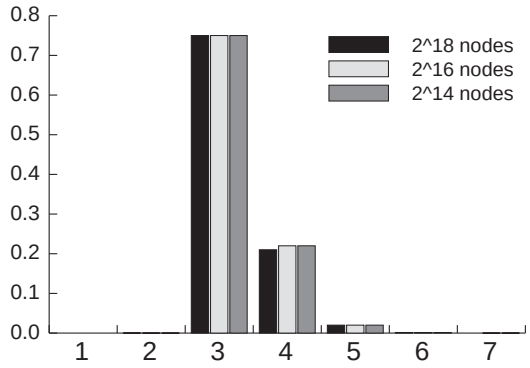


Figure 5.2: The distribution of the degrees when the maximum edge is taken.

walk ends at a node of degree d, it samples the node with probability $\frac{3}{d}$ (which is 1 most of the time) and initiates a new random walk with the remaining probability. The expected length of the walk increases by a factor of approximately 16/15.

The mixing time is estimated by calculating the second eigenvalue of the Laplacian. Figure 5.3 charts the spectral gap of the Laplacian as a function of the graph size. Each value is the average of 15 simulations. It could be seen that a spectral gap exists, and stands at about 0.256. The following theorem relates the spectral gap of a non-regular graph to the mixing time (see [31]). Denote by $\vec{\pi}$ the stationary distribution of the random walk. Denote by λ the second eigenvalue of the Laplacian matrix and by $d(x)$ the degree of node x.

Theorem 5.3.4. *For every graph with a normalized adjacency matrix $\hat{A}$ and any initial distribution $\vec{p}$,*

$$\max_x |(\hat{A}^t\vec{p})_x - \vec{\pi}_x| = ||\hat{A}^t\vec{p} - \vec{\pi}||_\infty \leq \frac{\max_x \sqrt{d(x)}}{\min_y \sqrt{d(y)}} \cdot e^{-t\lambda}$$

Plugging in a maximum degree of 6, minimum degree of 2 and $\lambda = 0.256$, we get that if we want each node to be sampled with probability within $\frac{1}{2n}$, the length of the walk should be at least $4.85 + 5.64 \log n$. This quantity must be further multiplied by $\frac{16}{15}$ to account for the extra walks needed for skewing the distribution to the uniform one. We conclude that the running time is about $5 + 5.75 \log n$. It should be noted that Theorem 5.3.4 is not tight, and a shorter walk would probably suffice. Indeed, we ran simulations and checked the variation distance between the random walk sample and the stationary distribution (which is *not* uniform, but which can be sampled to obtain a uniform distribution as discussed previously). The results are summarized is Table 5.3. A walk of length $5 \log n$ yields a variation distance of less than $\frac{1}{100}$ from stationary.

5.4 Applications

Consequences of expansion in skip graphs can be divided between those that use the expansions directly, like fault tolerance, and those that depend on the resulting rapid mixing of random walks. We discuss both below.

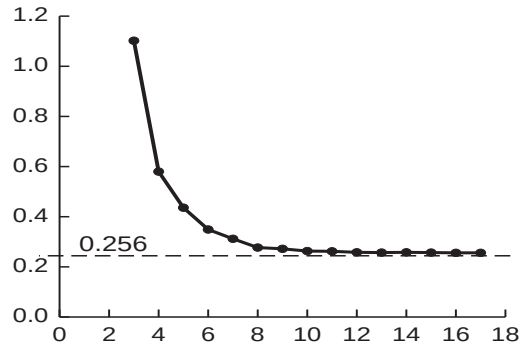


Figure 5.3: The spectral gap of the Laplacian as a function of $\log n$, using the maximum edge heuristic.

Walk Length	*variation distance*
$1 \log n$	0.72
$2 \log n$	0.29
$3 \log n$	0.095
$4 \log n$	0.025
$5 \log n$	0.007

Table 5.3: The variation distance between the random walk distribution and the stationary distribution, when $n = 2^{18}$,and the maximum edge is used.

5.4.1 Fault tolerance

Aspnes and Shah [16] showed via simulations that skip graphs are highly resilient to random failure of nodes. The expansion property of skip graphs gives the theoretical support to these empirical findings.

When adversarial faults are considered we have the following conclusion: If k nodes are deleted in an *adversarial* manner, then the largest connected component would have $\Omega(n - O(k))$ nodes. In other words, even if an adversary deletes a constant fraction of nodes, still a constant fraction of the nodes would remain connected.

5.4.2 Hitting large sets

As seen, a random walk of length $O(\log n)$ in an expander yields a random point. One of the most interesting and appealing properties of expander graphs is that in some sense a random walk of length $o(\log n)$ yields $\Theta(\log n)$ random samples. Clearly the nodes hit by the random walk are highly correlated, yet for the purpose of hitting a set of nodes, they behave as if they were independent random samples. The following theorem was proven by Ajtai, Komlós and Szemerédi [10]; it can also be found in Alon and Spencer's book [13].

Theorem 5.4.1 (Ajtai, Komlós and Szemerédi). *Let $G = (V, E)$ be a $d-$regular graph on n vertices, and suppose that each of its eigenvalues but the first one is at most λ. Let C be a set of*

cn vertices of G. Then, for every ℓ, the number of walks of length ℓ in G that avoid C does not exceed $(1-c)n((1-c)d+c\lambda)^\ell$.

There are nd^ℓ walks of length ℓ, which means that if we pick a random starting point for the walk, then the probability of avoiding the set C is at most:

$$\begin{aligned}&\frac{(1-c)n((1-c)d+c\lambda)^\ell}{nd^\ell}\\ =\ &(1-c)\cdot\left(\frac{(1-c)d+c\lambda}{d}\right)^\ell \le \left(\frac{(1-c)d+c\lambda}{d}\right)^\ell\end{aligned}$$

We know that a random walk of length $O(\log n)$ reaches a random point, therefore for any initial point, a random walk of length $\ell + O(\log n)$ avoids C with probability at most $\left(\frac{(1-c)d+c\lambda}{d}\right)^\ell = \left(1-(1-\frac{\lambda}{d})c\right)^\ell$. Note that the probability the set C is avoided by $\log n$ independent samples is $(1-c)^{\log n}$. The theorem states that this probability can be achieved by a walk of length $O(\log n)$.

We use the random-walk property in a somewhat different setting than the one described by Aspnes and Shah in [14]. We begin by briefly sketching this setting and then continue with applications of random walks.

5.4.3 The skip graph as a peer-to-peer data storage system

So far, we have assumed that each node of the graph represents a data item to be searched for. This allows for a simple implementation of *range queries* over the data set. Unfortunately, it also comes with a price: each data item is put separately in the system and requires roughly $O(\log n)$ communication links. Distributed Hash Tables which do not support range queries group elements together such that the total number of communication links is a function of the number of servers rather than data items. A natural way to group data items is to let each server hold a contiguous *segment* of the key space. Each server puts only one element of that segment in the skip graph (this data item effectively serves as the key of the skip graph node), and holds the rest in some internal data structure (which allows range queries). Now each node corresponds to a server in the system and not to a data item, and the capability to range query the data set is preserved.

Load balancing

The problem with the construction above is that it might be the case that different servers hold different fraction sizes of the data, thus dividing the load unevenly. A large fraction of the data set inflicts load not only due to the memory needed to store the items, but also because a server which holds many data items would be queried more often. It is therefore desirable that nodes share the data items as evenly as possible. Aspnes *et al* [15] and Karger and Ruhl [64] suggest various algorithms to deal with this problem. The algorithms they design apply a re-balancing mechanism that involves heavily loaded nodes actively seeking lightly loaded nodes. We take a different approach and suggest a simple *Join* algorithm, which preserves load balancing as long as the distribution from which data items receive their keys does not change often.

The algorithm we suggest is very simple. A server that wishes to join the skip graph performs a random walk of length $\Omega(\log n)$, recording the number of data items held by each node it encounters. It then picks a segment in the key space such that it splits the load of the most heavily loaded

of these nodes. Consider a set H of heavily loaded nodes. According to Theorem 5.4.1, the probability H is hit by a walk of length $\Omega(\log n)$ is equal to the probability it would be hit by $\Omega(\log n)$ *independent* random samples.

We simulated the second, simpler scenario. In our simulation we put 2^{27} data items in a single node, and then added 2^{18} nodes one by one. The kth node to enter sampled $\ln k$ nodes uniformly and independently and split the load of the heaviest sample. The resulting division of data items between the nodes was strikingly balanced. While the average load is 2^9, no node had a load larger than 2^{12}. Clearly this simulation oversimplifies the model: in particular, it does not take into account deletion of nodes and the dynamics of the data items themselves. Yet if we assume that the distribution of data item names is fixed, then it is reasonable to assume that the random walk would be a good load balancing heuristic. Testing the random walk algorithm in more realistic scenarios is an important future goal.

Locating highly replicated data items

Assume that some data item is immensely popular and appears at a large fraction of the nodes. Theorem 5.4.1 implies that a random walk would hit a node holding the popular item fast. It is shown in [78] that for popular data items an exhaustive search (as in Gnutella) is more efficient than an exact search using a DHT. It is shown in [48] that when the underlying graph is an expander, a random walk is more efficient than exhaustive search. We conclude that a skip graph may serve as an excellent *hybrid* data structure; i.e., may serve as a structured and unstructured P2P system simultaneously.

Gathering statistics

Assume we want to estimate what fraction of the skip graph nodes run Linux, or more generally we want to estimate the fraction of nodes which have some property. A natural approach would be to sample $\Theta(\log n)$ nodes randomly and use the sample to estimate the fraction. Typically a Chernoff bound is then used to show that the answer is approximately correct w.h.p. David Gillman [47] proved a theorem in the spirit of Theorem 5.4.1 and showed that one random walk of length $O(\log n)$ may serve to produce $\Theta(\log n)$ random samples. In the following we let λ denote the spectral gap of the normalized adjacency matrix (i.e. $1-\alpha$), let C be a set of cn nodes.

Theorem 5.4.2. *Let t_k be the number of visits to C in k steps of a random walk, starting from some arbitrary distribution, then* $\Pr[|t_k - ck| \geq \gamma] \leq ne^{-\frac{\gamma^2\lambda}{20k}}$.

Note that ck is the expected number of visits in the set. It follows that if c is a constant and k is $O(\log n)$ then w.h.p. the estimation of c could be arbitrarily close to c itself.

Chapter 6

Open Problems and Further Research

Clearly one of the most important tasks currently faced by the research community is to collect real data from actual implementations. In the theoretical aspect there are still many difficult and intriguing problems without a satisfactory solution. We conclude in this Chapter by outlining what we think are the main open problems that merit further research.

6.1 Problems Related to Load Balancing

Load-Balancing under Adversarial Deletions: The Continuous-Discrete approach relies on the equal division of the name space between the nodes (the *smoothness*). In Section 2.6 we presented several algorithms that guarantee smoothness. All of them but the *cyclic scheme* rely on the assumption that either there are no deletions; i.e. nodes never leave the system, or that nodes leave the system randomly. Theoretically speaking the cyclic scheme is appealing, as it offers excellent smoothness in the worst case even under adversarial insertions and deletions. Its main drawback is its vulnerability to concurrent joining and leaving. Say m nodes wish to leave the system in the same time. It is not clear how to coordinate them such that the smoothness property remains. A locking mechanism that would ensure that nodes leave (and join) one at a time implies a low throughput of the system and seriously limits its dynamic nature. Thus it seems that the cyclic scheme is less practical. It is important to come up with a scheme which is robust against adversarial insertions and deletions, yet is local enough to maintain high throughput. In particular, can the throughput of the cyclic scheme be increased?

Another limitation of the cyclic scheme is that it operates in a one dimensional name space only. As seen in Section 2.7, it is sometimes useful to have the names drawn from a two dimensional space. How can we modify the cyclic scheme to operate in a two dimensional space?

Exploiting Heterogeneity in DHT's: All the DHT's mentioned in this work were designed under the assumption that nodes in the network are identical – all have the same amount of resources. Clearly in the real world this is not the case. In the Internet some hosts have high bandwidth and some low, some allocate plenty of memory and some do not. It is desirable to allow each *physical node* of the network to fully utilize its resources, both for its own benefit and for the benefit of others.

A straight forward and well known approach (c.f. [63],[35]) for dealing with heterogeneity is to assign *several* (virtual) nodes to each physical node. In this scheme each physical entity simulates

many independent nodes, holding a different ID for each, and allocating separate resources for each simulation. The advantage of this approach is its simplicity and modularity. The network could be designed for homogenous nodes and the heterogeneity is taken into account only by the mapping of nodes to physical entities. The main drawback of this approach is that the overhead of maintaining and simulating several (virtual) nodes is replicated and multiplied. For instance a physical node simulating k virtual nodes will have to maintain k sets of overlay links. It is reasonable to assume that some of the replications could be saved. Indeed a different approach would be to deal with heterogeneity in the design of the virtual network itself. One way of doing this is by designing a Join protocol that assigns to each node a *contiguous* segment of the key space, with length *proportional* to the amount of resources the node has. The various algorithms presented in Section 2.6 aim to assign each node a segment of length $\frac{1}{n}$, and it is interesting to try and modify them so that they assign each node a different segment size. Godfrey and Stoica [50] take some steps in this direction.

Load Balancing Skip Graphs: In Section 5.4.3 we describe how to transform a skip graph into a data storage system. The main idea is to allow each server to hold a *segment* of the key space, and put in the skip graph only *one* data item which falls in the segment. Recall that in skip graphs data items may have arbitrary names, so we face the following load balancing problem: Given an arbitrary distribution of data item names in the ring $[0, 1)$, how to assign segments to servers such that each segment contains approximately the same number of data items? Simulations suggest that sampling $\log n$ random servers, and splitting the load of the heaviest server encountered is a good idea. It is desirable to improve the analysis and to find more clever algorithms for load balancing. Aspnes *et al* [15] made some progress in that direction.

6.2 Problems related to Security

Handling Byzantine Behavior in Network Construction: In Section 2.3 we handles the case where a random fraction of the nodes can send arbitrary messages relating to the content of data items. We named this adversarial behavior the *spam generating* model. The dynamic nature of P2P networks allows an adversary to do much more. A faulty node may influence the actual *topology* of the network and not merely the messages sent once the network exists. For instance a node may choose its ID such that it holds a specific piece of the key space, thus controlling all data items with keys that fall within that segment. A node may send false massages while executing the Join protocol (or as part of the maintenance protocol) causing its neighbors to have inconsistent views of the network, and so on. It would be interesting to find a method in which nodes could be *enforced* to behave according to protocol. A first step towards a secure system is the following: find a protocol that verifies that the ID of a joining node is chosen uniformly at random from the key space.

The egalitarian nature of a P2P network and the lack of central control means that it is not at all clear *who* should verify these things, and what actions should be done (and indeed by whom) once a foul play is detected. It is most likely that a combinatorial approach would not suffice and that cryptography would have to be used.

Pricing via Processing in a P2P setting: Even if all nodes adhere to the Join protocol, an adversary may repeatedly enter the system under various identities until it controls a specific portion of the network, what is known as the *Sybil attack* [38]. For instance, an adversary which

wants to control a specific data item may enter the system again and again until its random ID falls within a range that gives it control over that item. A possible direction towards dealing with the attack is to somehow enforce each joining node to solve some *moderately hard* problem (a-la the Pricing via Processing approach of Dwork and Naor [34]), thus forcing each joining node to 'pay' in computation. Hopefully such a scheme would imped Sybil attacks by making them too expensive for the adversary and thus uneconomical. As before, the lack of central control means that it is not clear who decides which function the joining node should compute and who verifies that the node indeed computed it. The idea has been implemented in a more limited setting by Maniatis*et al* [92].

Bibliography

[1] Amr El Abbadi, Dale Skeen, and Flaviu Cristian. An efficient, fault-tolerant protocal for replicated data management. In *Proceedings of the 5th ACM SIGACT/SIGMOD Conference on Principles of Database Systems*, pages 215–229, 1985.

[2] Ittai Abraham, Baruch Awerbuch, Yossi Azar, Yair Bartal, Dahlia Malkhi, and Elan Pavlov. A generic scheme for building overlay networks in adversarial scenarios. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, page 40, 2003.

[3] Ittai Abraham and Dahlia Malkhi. Probabilistic quorums for dynamic systmes. In *Proceedings of the 17th International Symposium on Distributed Computing (DISC)*, pages 60–74, 2003.

[4] Ittai Abraham, Dahlia Malkhi, and Gurmeet S. Manku. Brief announcement: Papillon: Greedy routing in rings. In *DISC*, 2005.

[5] Micah Adler, Eran Halperin, Richard M. Karp, and Vijay V. Vazirani. A stochastic process on the hypercube with applications to peer-to-peer networks. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing (STOC)*, pages 575–584, 2003.

[6] Divyakant Agrawal, Omer Egecioglu, and Amr El Abbadi. Billiard quorums on the grid. *Information Processing Letters*, 64(1):9–16, 1997.

[7] William Aiello, Baruch Awerbuch, Bruce M. Maggs, and Satish Rao. Approximate load balancing on dynamic and asynchronous networks. In *ACM Symposium on Theory of Computing*, pages 632–641, 1993.

[8] Michael Aizenman, Jennifer Chayes, Lincoln Chayes, Jurg Frohlich, and L. Russo. On a sharp transition from area low to perimeter low in a system of random surfaces. *Comm. Mathematical Physics*, (92):19–69, 1983.

[9] Michael Aizenman and Charles M. Newman. Discontinuity of the percolation density in one dimensional $1/|x-y|^2$ percolation models. *Communications in Mathematical Physics*, 107:611–647, 1986.

[10] Miklos Ajtai, János Komlós, and Endre Szemerédi. Deterministic simulation in LOGSPACE. In *Proceedings of the nineteenth annual ACM Symposium on Theory of Computing (STOC)*, pages 132–140, 1987.

[11] Noga Alon. Eigenvalues and expanders. *Combinatorica*, 6:83–96, 1986.

[12] Noga Alon and Vitali Milman. λ_1, isoperimetric inequalities for graphs and superconcentrators. *Journal of Combinatorial Theory*, 38:73–88, 1985.

[13] Noga Alon and Joel H. Spencer. *The Probabilistic Method*. John Wiley & Sons, second edition, 2000.

[14] James Aspnes, Zoë Diamadi, and Gauri Shah. Fault-tolerant routing in peer-to-peer systems. In *Proc. 21st ACM Symp. on Principles of Distributed Computing (PODC 2002)*, pages 223–232, July 2002.

[15] James Aspnes, Jonathan Kirsch, and Arvind Krishnamurthy. Load balancing and locality in range-queriable data. In *Twenty-Third ACM Symposium on Principles of Distributed Computing (PODC)*, pages 115–124, 2004.

[16] James Aspnes and Gauri Shah. Skip graphs. In *Proc. 14th ACM-SIAM Symp. on Discrete Algorithms (SODA 2003)*, pages 384–393, January 2003.

[17] James Aspnes and Udi Wieder. The expansion and mixing time of skip graphs with applications. In *Proc. 17th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*, pages 126–134, June 2005.

[18] Alberto László Barabási. *Linked: The New Science of Networks*. Perseus Publishing, 2002.

[19] Lali Barrière, Pierre Fraigniaud, E Kranakis, and D Krizanc. Efficient routing in networks with long range contacts. In *Proc. 15th Intl. Symp. on Distributed Computing (DISC 2001)*, pages 270–284, October 2001.

[20] Rida A. Bazzi. Planar quorums. In *Distributed Algorithms, 10th International Workshop, WDAG '96*, pages 251–268.

[21] Mark Bearden and Jr. Ronald P. Bianchini. A fault-tolerant algorithm for decentralized online quorum adaptation. In *Symposium on Fault-Tolerant Computing*, pages 262–271, 1998.

[22] Itai Benjamini and Noam Berger. The diameter of a longe-range percolation clusters on finite cycles. *Random Structures and Algorithms*, 19(2):102–111, 2001.

[23] Bittorrent. `http://www.bittorrent.com`.

[24] Bela Bollobas and Oliver Riordan. The critical probability for random voronoi percolation in the plane is 1/2. In *Front for the Mathematics ArXiv, math.PR/0410336*, 14 October 2004.

[25] John W. Byers, Jeffrey Considine, and Michael Mitzenmacher. Simple load balancing for distributed hash tables. In *Second International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 80–87, 2003.

[26] John W. Byers, Michael Luby, Michael Mitzenmacher, and Ashutosh Rege. A digital fountain approach to reliable distribution of bulk data. In *SIGCOMM*, pages 56–67, 1998.

[27] Jin-yi Cai. Essentially every unimodular matrix defines an expander. *Theory of Computing Systems*, 36(2):105–135, 2003.

[28] Miguel Castro, Peter Druschel, Y Charlie Hu, and Antony I T Rowstron. Topology-aware routing in structured peer-to-peer overlay networks. In *Proc. Intl. Workshop on Future Directions in Distrib. Computing (FuDiCo 2003)*, pages 103–107, 2003.

[29] Anawat Chankhunthod, Peter B. Danzig, Chuck Neerdaels, Michael F. Schwartz, and Kurt J. Worrell. A hierarchical Internet object cache. In *Proceedings of the USENIX 1996 annual technical conference*, pages 153–163.

[30] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathmematical Statistics*, 23:493–509, 1952.

[31] Fan R. K. Chung. Spectral graph theory. *Regional Conference Series in Mathematics, American Mathematical Society*, 92:1–212, 1997.

[32] Edith Cohen and Haim Kaplan. Balanced-replication algorithms for distribution trees. In *European Symposium on Algorithms (ESA)*, pages 297–309, 2002.

[33] Don Coppersmith, David Gamarnik, and Maxim Sviridenko. The diameter of a longe-range percolation graph. *Random Structures and Algorithms*, 21(1):1–13, 2002.

[34] Moni Naor Cynthia Dwork. Pricing via processing or combatting junk mail. In *The 12th Annual International Cryptology Conference (CRYPTO)*, pages 139–147, 1992.

[35] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, pages 202–215, 2001.

[36] Krzysztof Diks and Andrzej Pelc. Optimal adaptive broadcasting with a bounded fraction of faulty nodes. *Algorithmica*, 28(1):36–50, 2000.

[37] Peter Sheridan Dodds, Muhamad Roby, and Duncan J Watts. An experimental study of search in global social networks. *Science*, 301:827–829, 2003.

[38] John Douceur. The sybil attack. In *First International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 251–260, 2002.

[39] eMule. `http://www.emule-project.net`.

[40] Amos Fiat and Jared Saia. Censorship resistant peer-to-peer content addressable networks. In *Symposium on Discrete Algorithms (SODA)*, pages 94–103, 2002.

[41] Pierre Fraigniaud and Philippe Gauron. an overview of the content-addressable network d2b. In *Proc. 22rd ACM Symposium on Principles of Distributed Computing, (PODC)*, page 151, 2003.

[42] Pierre Fraigniaud, Cyril Gavoille, and Christophe Paul. Eclecticism shrinks even small worlds. In *Proc. 23rd ACM Symposium on Principles of Distributed Computing, (PODC)*.

[43] Joel Friedman. A proof of Alon's second eigenvalue conjecture. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing (STOC)*, pages 720–724, 2003.

[44] Ofer Gabber and Zvi Galil. Explicit construction of linear-sized superconcentrators. *Journal of Computer and System Science*, 22(3):407–420, 1981.

[45] Prasanna Ganesan and Gurmeet Singh Manku. Optimal routing in Chord. In *Proc. 15th ACM-SIAM Symp. on Discrete Algorithms (SODA 2004)*, pages 169–178, January 2004.

[46] Hector Garcia-Molina and Daniel Barbara. How to assign votes in a distributed system. *Journal of the Association for Computing Machinery*, 32(4):841–855, October 1985.

[47] David Gillman. A chernoff bound for random walks on expander graphs. *Siam Journal on Computing*, 27(4):1203–1220, 1998.

[48] Christos Gkantsidis, Milena Mihail, and Amin Saberi. Random walks in peer-to-peer networks. In *Proceedings of IEEE INFOCOM*, 2004.

[49] P. Brighten Godfrey, Karthik Lakshminarayanan, Sonesh Surana, Richard Karp, and Ion Stoica. Load balancing in dynamic structured P2P systems. In *Proc. IEEE INFOCOM*, Hong Kong, 2004.

[50] P. Brighten Godfrey and Ion Stoica. Heterogeneity and load balance in distributed hash tables. In *Proceedings of IEEE INFOCOM*, 2005.

[51] Oded Goldreich. *Randomized Methods in Computation - Lecture Notes*. http://www.wisdom.weizmann.ac.il/∼oded/rnd.html, 2001.

[52] Geoffrey Grimmett. *Percolation*. Springer-Verlag, second edition, 1999.

[53] Krishna P Gummadi, Ramakrishna Gummadi, Steven D Gribble, Sylvia Ratnasamy, Scott Shenker, and Ion Stoica. The impact of DHT routing geometry on resilience and proximity. In *Proc. ACM SIGCOMM 2003*, pages 381–394, 2003.

[54] Zygmunt J. Haas and Ben Liang. Ad hoc mobility management with uniform quorum systems. *IEEE/ACM Transactions on Networking*, 7(2):228–240, 1999.

[55] Torben Hagerup and Christine Rüb. A guided tour of Chernoff bounds. *Information Processing Letters*, 33(6):305–308, 1990.

[56] Nicholas Harvey and J Ian Munro. (brief announcement) deterministic Skipnet. In *Proc 22nd ACM Symposium on Principles of Distributed Computing (PODC 2003)*, pages 152–153, 2003.

[57] Nicholas J A Harvey, Michael Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. Skipnet: A scalable overlay network with practical locality properties. In *Proc. 4th USENIX Symposium on Internet Technologies and Systems (USITS 2003)*, 2003.

[58] Yehuda Hassin and David Peleg. Average probe complexity in quorum systems. In *20th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 180–189, 2001.

[59] Maurice Herlihy. Dynamic quorum adjustment for partitioned data. *ACM Transactions on Database Systems (TODS)*, 12:170–194, 1987.

[60] Kirsten Hildrum and John Kubiatowicz. Asymptotically efficient approaches to fault-tolerance in peer-to-peer networks. In *DISC*, pages 321–336, 2003.

[61] Sushil Jajodia and David Mutchler. Dynamic voting algorithms for maintaining the consistency of a replicated database. *ACM Transactions on Database Systems*, 15(2):230–280, June 1990.

[62] M. Frans Kaashoek and David R. Karger. Koorde: A simple degree-optimal distributed hash table. In *Second International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 98–107, 2003.

[63] David Karger, Eric Lehman, F.Thomson Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *ACM Symposium on Theory of Computing (STOC)*, pages 654–663, 1997.

[64] David R. Karger and Matthias Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. In *Proceedings ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 36–43, 2004.

[65] Goutham Karumanchi, Srinivasan Muralidharan, and Ravi Prakash. Information dissemination in partitionable mobile ad hoc networks. In *Proceedings of IEEE Symposium on Reliable Distributed Systems*, pages 4–13, 1999.

[66] Krishnaram Kenthapadi and Gurmeet S. Manku. Decentralized algorithms using both local and random probes for p2p load balancing. In *Proceedings ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 135–144, 2005.

[67] Valerie King, Scott Lewis, and Jared Saia. On algorithms for choosing a random peer. Unpublished manuscript, 2005.

[68] Valerie King and Jared Saia. Choosing a random peer. In *Proceedings of the 23nd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 125–130, 2004.

[69] Jon Kleinberg. Navigation in a small world. *Nature*, 406:845, 2000.

[70] Jon Kleinberg. The small-world phenomenon: An algorithmic perspective. In *Proc. 32nd ACM Symposium on Theory of Computing (STOC 2000)*, pages 163–170, 2000.

[71] Richard R. Koch, F. Thomson Leighton, Bruce M. Maggs, Satish Rao, Arnold L. Rosenberg, and Eric J. Schwabe. Work-preserving emulations of fixed-connection networks. *Journal of the ACM*, 44(1):104–147, January 1997.

[72] Michael Larsen. Navigating the cayley graph of $\mathrm{SL}_2(\mathbb{F}_p)$. *International Mathematics Research Notices*, 27:1465–1471, 2003.

[73] Ching Law and Kai-Yeung Siu. Distributed construction of random expander graphs. In *Proceedings of IEEE INFOCOM*.

[74] Emmanuelle Lebhar and Nicolas Schabanel. Almost optimal decentralized routing in long-range contact networks. In *Proceedings of th 31st International Colloquium on Automata, Languages and Programming (ICALP)*, pages 894–905, 2004.

[75] F. Thomson Leighton. *Introduction to parallel algorithms and architectures : arrays, trees, hypercubes*, chapter 3.3. Morgan Kaufmann, San Mateo, CA, 1992.

[76] F. Thomson Leighton and Bruce Maggs. Expanders might be practical: Fast algorithms for routing around faults in multibutterflies. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 384–389, 1989.

[77] Thomas L. Liggett, Roberto H. Schonmann, and Alan M. Stacey. Domination by product measures. *The Annals of Probability*, 25(1):71–95, January 1997.

[78] Boon Thau Loo, Ryan Huebsch, Ion Stoica, and Joseph M. Hellerstein. The case for a hybrid p2p search infrastructure. In *3rd International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 141–150, 2004.

[79] Esti Yeger Lotem, Idit Keidar, and Danny Dolev. Dynamic voting for consistent primary components. In *Symposium on Principles of Distributed Computing (PODC)*, pages 63–71, 1997.

[80] Nancy Lynch and Alexander Shvartsman. Robust emulation of shared memory using dynamic quorum-acknowledged broadcasts. In *Symposium on Fault-Tolerant Computing*, pages 272–281, 1997.

[81] Nancy Lynch and Alexander Shvartsman. Rambo: A reconfigurable atomic memory service for dynamic networks. In *Proceedings of the 16th International Symposium on Distributed Computing*, pages 173–190, 2002.

[82] Mamoru Maekawa. A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems. *ACM Transactions on Computer Systems*, 3(2):145–159, May 1985.

[83] Bruce M. Maggs and Eric J. Schwabe. Real-time emulations of bounded-degree networks. *Information Processing Letters*, 66(5):269–276, 1998.

[84] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proc. 21rd ACM Symposium on Principles of Distributed Computing, (PODC)*, pages 183–192, 2002.

[85] Gurmeet S. Manku. Balanced binary trees for id management and load balance in distributed hash tables. In *Proc. 23rd ACM Symposium on Principles of Distributed Computing, (PODC)*, pages 197–205, 2004.

[86] Gurmeet Singh Manku. Routing networks for distributed hash tables. In *Proc. 22nd ACM Symp. on Principles of Distributed Computing (PODC 2003)*, pages 133–142, July 2003.

[87] Gurmeet Singh Manku, Mayank Bawa, and Prabhakar Raghavan. Symphony: Distributed hashing in a small world. In *Proc. 4th USENIX Symposium on Internet Technologies and Systems (USITS 2003)*, pages 127–140, 2003.

[88] Gurmeet Singh Manku, Moni Naor, and Udi Wieder. Know thy neighbour's neighbour: The role of lookahead in randomized P2P networks. In *Proc. 36th ACM Symposium on Theory of Computing (STOC 2004)*, pages 54–63, 2004.

[89] G. A. Margulis. Explicit constructions of concentrators. *Problemy Peredachi Informatsii*, (9(4)), October - December 1973.

[90] Charles Martel and Van Nguyen. Analyzing kleinberg's (and other) small-world models. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing (PODC)*, pages 179–188, 2004.

[91] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *The First International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 53–65, 2002.

[92] Petros Maniatis, Mema Roussopoulos, T J Giuli, David S. H. Rosenthal, and Mary Baker The LOCKSS Peer-to-Peer Digital Preservation System. *ACM Transactions on Computer Systems (TOCS)*, 23(1):2–50, 2005.

[93] Mikhail V. Menshikov. Coincidence of critical points in percolation problems. *Soviet Mathematics Doklady*, 33:856–859, 1986.

[94] Stanley Milgram. The small world problem. *Psychology Today*, 67(1):60–67, May 1967.

[95] Yaron Minsky and Ari Trachtenberg. Practical set reconciliation. Technical Report 2002-03., Boston University, 2002.

[96] Michael Mitzenmacher, Andrea Richa, and Ramesh Sitaraman. The power of two random choices: A survey of the techniques and results. In *Handbook of Randomized Computing, P. Pardalos, S. Rajasekaran, and J. Rolim, Eds. Kluwer.*, 2000.

[97] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1997.

[98] Uri Nadav and Moni Naor. Fault-tolerant storage in a dynamic environment. In *Proceedings of the 18th Annual Conference on Distributed Computing (DISC), Lecture Notes in Computer Science 3274*, pages 390–404. Springer, 2004.

[99] Moni Naor and Udi Wieder. Novel architectures for P2P applications: The continuous-discrete approach. In *Proc. 15th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA 2003)*, pages 50–59, June 2003.

[100] Moni Naor and Udi Wieder. Scalable and dynamic quorum systems. In *Proc. 22st ACM Symp. on Principles of Distributed Computing (PODC 2003)*, pages 114–122, 2003.

[101] Moni Naor and Udi Wieder. A simple fault tolerant distributed hash table. In *Second International Workshop on Peer-to-Peer Systems*, pages 88–97, February 2003.

[102] Moni Naor and Udi Wieder. Know thy neighbor's neighbor: Better routing for skip-graphs and small worlds. In *The Third International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 269–277, 2004.

[103] Moni Naor and Udi Wieder. Scalable and dynamic quorum systems. *Distributed Computing, PODC 2003 Special Issue*, 17(4):311–322, May 2005.

[104] Moni Naor and Avishai Wool. Access control and signatures via quorum secret sharing. *IEEE Transactions on Parallel and Distributed Systems*, 9(9):909–922, 1998.

[105] Moni Naor and Avishai Wool. The load, capacity, and availability of quorum systems. *SIAM Journal on Computing*, 27(2):423–447, 1998.

[106] Charles M Newman and Lawrence 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.

[107] Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, and Song N. Chiu. *Spatial Tessellations — Concepts and Applications of Voronoi Diagrams*. Wiley, Chichester, second edition, 2000.

[108] David Peleg and Avishai Wool. How to be an efficient snoop, or the probe complexity of quorum systems. *SIAM Journal on Discrete Mathematics*, 15(3):416–433, August 2002.

[109] C. Greg Plaxton and Rajmohan Rajaraman. Fast fault-tolerant concurrent access to shared objects. In *IEEE Symposium on Foundations of Computer Science*, pages 570–579, 1996.

[110] Ithiel Pool and Manfred Kochen. Contacts and influence. *Social Networks*, 1:1–48, 1978.

[111] Roberto De Prisco, Alan Fekete, Nancy Lynch, and Alexander Shvartsman. A dynamic view-oriented group communication service. In *Symposium on Principles of Distributed Computing (PODC)*, pages 227–236, 1998.

[112] William Pugh. Skip lists: A probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, June 1990.

[113] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. In *Proc ACM SIGCOMM*, pages 161–172, San Diego CA, Augost 2001.

[114] Sylvia Ratnasamy, Scott Shenker, and Ion Stoica. Routing algorithms for dhts: Some open questions. In *First International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 45–52, 2002.

[115] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.

[116] Jared Saia, Amos Fiat, Steven Gribble, Anna R. Karlin, and Stefan Saroiu. Dynamically fault-tolerant content addressable networks. In *First International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 270–279, 2002.

[117] Beverly A. Sanders. The information structure of distributed mutual exclusion algorithms. *ACM Transactions on Computer Systems*, 5(3):284–299, August 1987.

[118] Lawrence S. Schulman. Long range percolation in one dimension. *Journal of Physics A*, 16(17):L639–L641, 1983.

[119] Skype. `http://www.skype.com`.

[120] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[121] Ivan Stojmenović and Pedro E. V. Pena. A scalable quorum based location update scheme for routing in ad hoc wireless networks. Technical Report TR-99-09, SITE, University of Ottawa, September 1999.

[122] Leslie G. Valiant. A scheme for fast parallel communication. *SIAM Journal on Computing*, 11(2):350–361, May 1982.

[123] Duncan J. Watts and Steven Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

[124] Hakim Weatherspoon and John Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *First International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 328–338, 2002.

[125] Jun Xu, Abhishek Kumar, and Xingxing Yu. On the fundamental tradeoffs between routing table size and network diameter in peer-to-peer networks. In *INFOCOM*, 2003.

[126] Kevin C. Zatloukal and Nicholas J. A. Harvey. Family trees: an ordered dictionary with optimal congestion, locality, degree, and search time. In *Proc. 14th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 308–317, 2004.

[127] Hui Zhang, Ashish Goel, and Ramesh Govindan. Incrementally improving lookup latency in distributed hash table systems. In *ACM SIGMETRICS 2003*, pages 114–125, June 2003.

[128] Ben Y. Zhao and John Kubiatowicz. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB CSD 01-1141, University of California at Berkeley, Computer Science Department, 2001.