THE DESIGN AND IMPLEMENTATION OF A ROBUST, COST-CONSCIOUS PEER-TO-PEER LOOKUP SERVICE

A Thesis
Presented to
The Academic Faculty

by

Cyrus M. Harvesf

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
December 2008

THE DESIGN AND IMPLEMENTATION OF A ROBUST, COST-CONSCIOUS PEER-TO-PEER LOOKUP SERVICE

Approved by:

Professor Douglas M. Blough,
Committee Chair
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor George F. Riley
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Douglas M. Blough, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Sudhakar Yalamanchili
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Henry L. Owen
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Ling Liu
College of Computing
Georgia Institute of Technology

Date Approved: 14 November 2008

In loving memory of my father, the first Dr. Harvesf

ACKNOWLEDGEMENTS

It takes a village to raise a child.

-African Proverb

My life has been touched by a village of wonderful people. First, I would like to thank my advisor, Doug Blough. I appreciate your guidance, leadership, motivation and understanding throughout my education. It has been an honor and privilege to work beside you for nearly six years. I will always hold our friendship near to my heart.

I would also like to extend warm thanks to the members of my dissertation committee: Ling Liu, Henry Owen, George Riley, and Sudhakar Yalamanchili. I appreciate your guidance throughout the dissertation process. I would like to offer a special thanks to George Riley for all of the valuable discussions we shared.

Next, I would like to thank all of my colleagues with whom I shared academic discussion, light-hearted banter, late nights, and a delightful four years. I would like to give special thanks to Revathi Balakrishnan, David Bauer, Venkata Siva Vijayendra Bhamidipati (aka Vijay), Raj Bhattacharjea, Luis Miguel Cortés Peña, Elizabeth Lynch, Apurva Mohan, Kulin Shah, Ramya Srinivasan, and Mike Sun with whom I've developed enduring friendships.

I've also been blessed with a group of close friends, beginning with Karthik Balakrishnan. Our friendship has endured tens of thousands of night and weekend minutes. I enjoyed every single one. I look forward to many afternoons on the fairways with you in the near future. To Neil Joshi and Juan Mojica, thank you for our annual "reunions" and for all the laughs we've shared. I look forward to many more. To

Emmalee Nichols, thank you for all the great times and for being there when I needed you most. You will always be a dear friend. To Elizabeth and Eric Lynch, I enjoyed all our shared lunches, dinners, and laughs. I cherish our friendship and thank you for all the times you were there for me with Tylenol, warm chocolate muffins, and smiles. To Jin Joo Lee, I wish I had met you sooner, but I am thankful that our paths in life crossed. You can always count on your "big brother".

Finally, to my family, those here and those who have gone on, I cannot express my thanks. I can only say what I fail to say enough: I love you. To my uncle, my father could not have asked for a better brother and I could not have asked for a better uncle. I love you. To my sister, from the days when you "tickled me to death" to all the care packages you sent me that brightened my days, you've always been there for me. I love you. To my father, thank you for instilling in me the values that made you the great man you were. I love you and miss you dearly.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Lookup services form a core component in a number of applications such as name resolution, many publish-subscribe systems, and IP communication systems. With the rapid growth of the Internet, lookup services must be designed with scalability and resilience in mind. The peer-to-peer (p2p) architecture exhibits these and other desirable qualities that make it an excellent substrate for lookup applications. However, p2p is not widely used for critical lookup infrastructure like the domain name system (DNS) because of its limited dependability and security. This dissertation aims to address its shortcomings through the design and implementation of a robust, cost-conscious p2p lookup service.

The distributed nature of p2p makes it difficult to maintain dependability and security. Hundreds of thousands, if not millions, of nodes located worldwide on hardware of varying capabilities may be responsible for storing objects, servicing lookup requests, and forwarding overlay messages. Failure, corruption or compromise of a small set of these nodes could have a dramatic effect on the ability of the system to resolve lookup requests.

We employ replica placement to mitigate the impact of failed nodes on overlay connectivity and the availability of objects in a lookup service. We demonstrate how an appropriate placement improves object availability and node reachability by creating route diversity. We show how equally-spaced replica placement can create provably disjoint routes in a well known distributed hash table implementation called Chord. Through detailed simulation, we show that this placement creates greater route diversity than existing placements. Furthermore, we show that the number of disjoint routes has a pronounced impact on the reachability of an object replica

and, in turn, the probability of lookup success. We generalize this approach into MaxDisjoint, a replica placement for a broader class of distributed hash table implementations that employ a routing scheme we call tree-based routing. Using Pastry as example, we show that MaxDisjoint performs comparably to equally-spaced placement in Chord and can dramatically improve the probability of lookup success for a wider range of DHT implementations.

If the dependability and security needs of a p2p application can be met using replica placement and other existing mechanisms, then there is an opportunity to make p2p pervasive in practice. However, this requires that p2p designs become more aware of their operating environments. The illegal file-sharing beginnings of p2p have made most advances in the field "client-driven"; that is, enhancements to the architecture are made primarily to improve user experience (e.g., lookup latency, availability). As such, most existing p2p applications are agnostic to the needs of the underlying network operators. Messages may be routed over tangled routes across numerous Internet Service Providers (ISPs) without regard to the IP transit costs incurred. This becomes increasingly expensive for ISPs with the transfer of large files. This dissertation proposes an id assignment policy to bound the costs of overlay routes and dramatically reduce costs incurred by a p2p lookup service deployed across several ISPs. Combining replica placement with this id assignment, further cost benefits can be achieved; however, there is a trade-off between the cost of replication and transit costs. We propose a function to quantify the cost of replica storage and replica retrieval that can be used to find the optimal replication degree. We evaluate this technique using an experimental deployment of our lookup service implementation on the Emulab testbed and find that it can reduce network operating costs by as much as 80%.

By improving the dependability, security and operating cost of a p2p lookup service, this dissertation seeks to unlock the potential of p2p technology for legitimate

applications. Dependable, secure and cost-effective p2p is a natural platform for delivering rich content like data, voice and streaming video in a fashion that is scalable to the demands of the Internet.

CHAPTER I

INTRODUCTION

The Internet has become the ultimate platform for the exchange of data, voice and video. As the Internet grows, content distribution networks are challenged with the cost of delivering rich media to the masses. The classical client-server networking paradigm is becoming overwhelmed with the demand that the Internet has created. Thus, we have turned to alternative architectures to provide the scalability and resilience necessary to deploy these applications more effectively. The peer-to-peer (p2p) paradigm is one such solution.

In the p2p architecture, each peer is able to act as a client and a server. Thus, the workload can be balanced across a large number of peers. Bittorrent (data), Skype (voice) and Joost (video) have all enjoyed success as p2p content distributors, but there is reluctance to deliver critical services via p2p. In general, as the sensitivity of the shared media increases, so does the reluctance to use p2p. Since p2p distributes responsibility among hundreds of thousands or millions of nodes, it is not straightforward to maintain the availability, integrity and confidentiality of the service. In the first part of this dissertation, we focus on the problem of availability and present our solution as a robust p2p lookup service where object availability is maintained in the presence of failures or malicious behavior.

A second impediment to the deployment of p2p technology for popular services is the cost incurred by network operators. Most p2p networks rely on multi-hop routing through several peers, which may reside in multiple Internet Service Provider (ISP) domains. ISPs rely on each other to forward data to other portions of the Internet, sometimes with a non-zero IP transit cost. Since each hop in a p2p route may reside

in a different ISP, the multi-hop p2p route between two peers may incur several times the cost of the direct Internet route between the source and destination. To address this problem, we integrate a cost-conscious mechanism into our lookup service design and show through experimentation that it reduces ISP costs by 80%.

In the remaining sections of this chapter, we define a lookup service and its potential applications, introduce the concept of a p2p overlay, discuss the shortcomings of overlay routing, enumerate our contributions and lay out the organization of the remainder of the dissertation.

1.1 Lookup Services

A lookup service provides two basic operations: *insert* and *lookup*. When an object is inserted, it is persistently stored and associated with some metadata. To perform a lookup, a request is made to the service using the metadata associated with the desired object. Lookup services form a core component in many common applications such as name resolution, publish-subscribe, and IP communication systems.

Domain name resolution is a classic lookup service example. Network addresses are stored in the system and associated with a domain name. When a client needs to resolve a domain name, the service is queried using the domain name and the associated network address is returned. Publish-subscribe systems behave in the same manner; messages are associated with a "topic" on insertion and subscribers look up a particular topic to fetch the published messages.

Because it is a versatile component central to many applications, we use a lookup service as the vehicle for the contributions of this dissertation. As a result, the benefits of our findings can be far-reaching and impact a large number of services.

1.2 Peer-to-Peer Overlays

A central concept of this work is the notion of an *overlay*. Although they are often referred to as "peer-to-peer networks", it is important to realize that these are logical

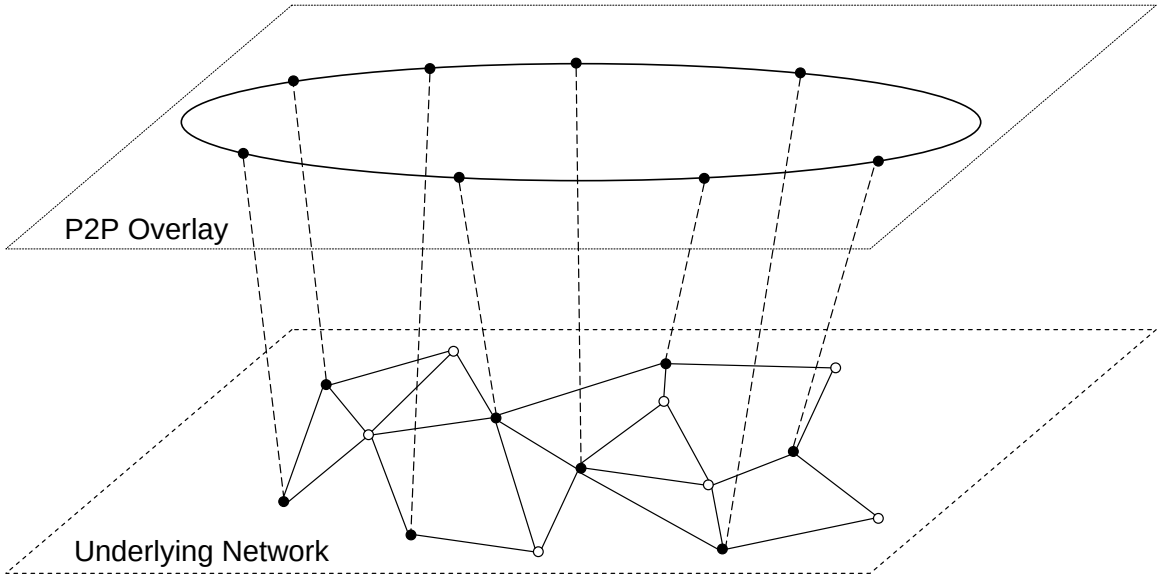


Figure 1: The relationship between a p2p logical ring overlay and the underlying network. The dashed lines connect nodes in the underlying network to their logical location in the overlay.

networks *overlaid* on an existing network. The relationship between the p2p overlay and the underlying network is depicted in Figure 1. Note that although two overlay nodes may be direct logical neighbors in the logical ring topology, they may be separated by several hops in the underlying network. In this dissertation, we will use the term overlay or overlay network to refer to the logical peer-to-peer topology.

For the purposes of this dissertation, we will assume that the overlay is comprised of end hosts in the Internet. Furthermore, we assume that the underlying network eventually delivers any message sent from one overlay node to another. We feel this assumption is reasonable since the Internet path between two overlay nodes is comprised primarily of Internet routers. Given these assumptions, we focus primarily on addressing failed and compromised nodes in the overlay and the economic impact of overlay functions on the underlying network.

1.3 Shortcomings of Structured Overlay Routing

Before summarizing the contributions of the dissertation, we will provide more detail of the problems that are addressed. Both are related to the shortcomings of structured overlay routing. First, we address the problem of object availability in a p2p lookup service. We argue that replication alone is not sufficient to address object availability in structured p2p overlays. Since objects are fetched along multi-hop routes, some of the replicas may become unreachable if the routes are compromised. The system is more vulnerable when the routes to a replica set overlap creating a single point of failure. Therefore, we use replica *placement* to create route diversity in the paths used to fetch a replica set.

Second, overlay routing can be quite efficient with respect to path lengths in the overlay. Unfortunately, each hop in the overlay may correspond to a multi-hop path in the underlying network, which has many undesirable side effects. The relationship between an overlay route and the route in the underlying network is shown in Figure 2. In this simple example, an overlay route of 3 hops corresponds to a path in the underlying network of 7 hops. Besides the impact this may have on latency, this may result in significant IP transit costs if the hops in the underlying network lay in commercially diverse networks. Innovation in overlay routing has addressed the issue of long path lengths with distributed hash tables (DHTs) [31, 32, 39, 43] by structuring the p2p topology to bound the number of overlay routing hops. We turn our attention to the economic impact of overlay routing.

The economic burden placed on network operators threatens the survival of p2p technology. Consider as a worst case scenario an overlay route between two peers that reside in the same network. Suppose that this route takes multiple hops in the overlay from the source node through networks all over the globe before arriving at the destination node. Traversing these intermediate networks may incur transit costs for their operators. Most p2p overlay designs are agnostic to the underlying network and

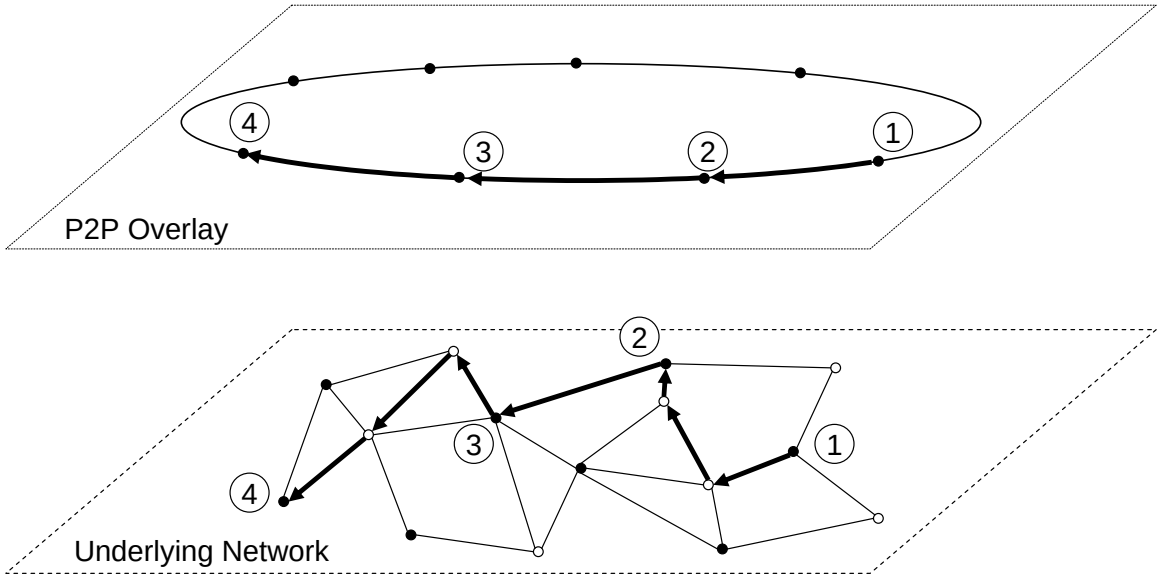


Figure 2: The relationship between an overlay route and the route in the underlying network.

do not address transit costs. We propose a p2p overlay design that employs replica placement to provide cost benefit to the underlying network operators in addition to improving object availability in the overlay.

1.4 Contributions

The goal of this research is to design and implement a p2p lookup service that can be used to provide critical services robustly without placing an excessive economic burden on network operators. Contributions made as part of this research are as follows:

- To improve routing robustness in Chord [39], a well known DHT, we propose equally-spaced replica placement. We show that this placement creates provably disjoint routes from any source node to the replica set [21]. Furthermore, we provide a theorem to find the replication degree necessary to create a desired number of disjoint routes. Through simulation, we show that the number of disjoint routes has a direct impact on routing robustness, which we measure as the probability of lookup success.

- To extend the benefits of equally-spaced replication to DHTs other than Chord, we generalize the approach for DHTs that employ a tree-based routing scheme. We propose MaxDisjoint, a replica placement for tree-based routing DHTs, and prove that it creates disjoint routes from any source node to the replica set. Furthermore, we determine the necessary replication degree to create a desired number of disjoint routes. Using simulations of Pastry, we evaluate MaxDisjoint and verify that it maintains the benefits of equally-spaced replica placement.

- Although replication can provide many benefits, fetching an entire replica set as opposed to a single objects creates additional traffic in the p2p overlay. To measure the impact of parallelization on overlay congestion and, in turn, lookup response time, we considered a family of replica fetching strategies that vary the degree of parallelization to reduce congestion and maintain a reasonable response time. We evaluate these techniques to find the strategy that best trades off load and latency.

- To demonstrate the workings of MaxDisjoint and the ease with which it can be integrated into an existing DHT, we implemented a prototype using the FreePastry distribution from Rice University [1]. We use this prototype to produce many of our experimental results.

- To address the transit costs incurred by overlay routes, we describe organization-based node id assignment for DHTs that employ a prefix-matching routing scheme. We prove that overlay paths in a DHT with organization-based id assignment incur a bounded transit cost. We show experimentally that this cost is significantly less (55-65%) than that incurred by existing DHT designs.

- To reduce cost further, we propose using replication to increase the availability

of a zero transit cost replica. We show that replica placements like MaxDisjoint can also provide a number of bounded cost paths in the event that a zero transit cost replica is not available. We define a cost function that can be used to find the optimal replication degree, which minimizes transit costs without overburdening networks with the storage of objects. Through experimentation, we show that using MaxDisjoint in conjunction with organization-based id assignment can result in an 80% reduction in cost.

1.5 Dissertation Organization

This dissertation is organized as follows:

Chapter 2 reviews other work on peer-to-peer routing, replica placement, and cost-conscious overlays.

Chapter 3 provides necessary background on distributed hash tables.

Chapter 4 describes equally-spaced replication and analyzes the impact of replica placement on routing robustness in Chord.

In Chapter 5, we introduce tree-based routing DHTs, the MaxDisjoint replica placement, and evaluate MaxDisjoint in Pastry.

Chapter 6 proposes a cost-conscious distributed hash table design for Internet service providers. Organization-based id assignment is defined and its cost benefits with MaxDisjoint are discussed.

Finally, in Chapter 7, we provide concluding remarks and some direction for future work.

CHAPTER II

RELATED WORK

The contributions of this dissertation address the dependability shortcomings and economic needs of peer-to-peer overlays. Ours is the first approach to use replica placement to mitigate these two seemingly unrelated problems. Because of the novelty of our approach, there is little closely related previous work. Nevertheless, there is a myriad of work that have considered the two problems independently. In this chapter, we will first review the threats to overlay security and discuss existing solutions to mitigate their effects. In the remaining sections, we will contrast our contributions with existing node id assignment policies, replica placements, and cost-conscious overlays.

2.1 Structured Peer-to-Peer Overlay Security

The body of work in overlay security is extensive and, although our work focuses on overlay routing, we feel it is worthwhile to briefly survey some of the other relevant contributions before detailing routing security literature. We categorize overlay security works according to the type of attacks they address. Sit and Morris [36] classify attacks on distributed hash tables into three categories:

1. Storage and retrieval attacks, which target the manner in which peers manage data items;

2. Routing attacks, which target the manner in which peers route messages; and

3. Miscellaneous attacks, which target other aspects of the system, such as admission control or the underlying network routing service.

Although this categorization focuses on DHTs, it is applicable to peer-to-peer overlays in general. All peer-to-peer overlays perform some form of routing and, when used to implement lookup services, they rely heavily on the storage and retrieval of data items.

Sit and Morris point to replication as the primary method of mitigating the effects of storage and retrieval attacks [36]. With multiple copies of a data item, we increase the likelihood that a correct node is responsible for a given key. Typically, replicas are placed in the neighborhood of the master key; this is the case in the original Chord [39], Pastry [32], and CAN [31] papers. We will show that such placements are not sufficient to address the storage and retrieval problem; the retrieval of objects in a peer-to-peer overlay is a complex operation that requires the support of a robust routing infrastructure. We will detail some alternative placements in Section 2.3.

The body of work related to miscellaneous attacks is too extensive to carefully consider here, but we highlight a few well known problems nonetheless. Many works have focused on admission control problems [33], the Sybil [12] and Eclipse [35] attacks in particular, and denial of service (DoS) attacks [14, 38]. These types of attacks may affect a broad range of DHTs or may be specific to a particular implementation. Considering these types of work further is beyond the scope of this dissertation.

Our work focuses on limiting the effects of routing attacks and the success of storage and retrieval attacks. Although several solutions suggest using cryptographic signatures [7, 23, 33, 36, 40], such an approach typically requires a central trusted authority to issue certificates to newly joining nodes. This centralized entity diminishes the benefits of a fully distributed approach. It is preferable to design robustness into the distributed routing infrastructure to create diverse routing paths [4, 7, 30, 37]. Multiple, diverse routes increase the likelihood that failed or malicious peers can be circumnavigated.

We define a set of two or more routes to be *diverse* if and only if every pair of routes differs in at least one hop. Typically, a set of diverse routes share a common source node. Portmann et al. [30] consider using the routing table entries of the source node to create diverse routes. They empirically determine the Chord finger table entries that minimize route overlap when used as the first hop. We will show that replica placement can be used to create provably disjoint routes, which share no overlap.

Srivatsa and Liu [37] propose the notion of *independent routes* to be a set of two or more routes where every pair of routes differs in all hops but the source and destination. Srivatsa and Liu use the neighbors of the source node to create independent routes to a destination. Although routing via neighbors of the source node creates some route diversity, we will show that it is insufficient to dramatically improve routing robustness.

Cyclone [4] is an equivalence-based routing scheme built upon an existing DHT that also aims to create route diversity. Independent routes are created by routing along different equivalence classes. This approach incurs additional per node routing overhead and involves a complex routing scheme. Furthermore, independent routes address routing attacks, but fail to fully address the storage and retrieval problem. Without replication, the storage and retrieval attack is as simple as compromising the node responsible for the lone copy of the victim object. Our approach of creating disjoint routes through replication addresses both problems.

Other routing solutions have addressed routing attacks without making route diversity the focus of the approach. Castro et al. [7], for example, have developed a secure routing primitive, which guarantees that when a non-faulty node sends a message to a key k, the message is delivered to all non-faulty members of the replica set of k. This holistic solution combines secure node id assignment, improved routing table maintenance using constrained routing tables, routing fault detection techniques,

and a neighbor set anycast to support the secure routing primitive. The approach is sophisticated and creates excellent results. Although the approach is not as strongly reliant on route diversity as the others discussed above, it uses the neighbor set to create diverse routes similar to the approach used in [37]. Unfortunately, it relies heavily on the presence of certified node ids and a central certifying authority. Cryptographic techniques may place strain on nodes with limited computing power and a central certifying authority presents a point of vulnerability in the system. Furthermore, the approach requires modification of the existing DHT routing mechanisms, which makes it cumbersome to transfer its benefits from one DHT to another. Our approach can be deployed as a middleware on top of an existing routing infrastructure without the need for a central certifying authority or certified node ids.

Mickens and Noble [27] develop a framework for diagnosing broken overlay routes, whether they result from IP-level link failures or malicious routers at the overlay-level. Once the source of the error is detected, DHT routes can circumnavigate broken IP-level links or exclude misbehaving nodes to improve the routing success rate of the system. Our work does not consider the failure of IP-level links; however, disjoint overlay routes may also be used to circumnavigate broken links without excluding any overlay nodes. We believe that circumnavigating failures is preferred to the exclusion of overlay nodes. Each overlay node represents a unit of resources available to the entire network. As nodes are blacklisted, the resources of the entire network are diminished.

It is worth noting that path disjointedness has been considered in other contexts as well. Castro et al. [8] propose a p2p multicast system for high bandwidth content (e.g., streaming video). Content is split into "stripes" such that the quality of the content improves with the number of stripes downloaded simultaneously. The stripes are delivered to subscribers via multicast trees. To ensure that the failure of a single node does not compromise all stripes, the trees are constructed to be inner node

disjoint. In other words, no node will be an inner node of more than one multicast tree. Therefore, if a single node fails, no more than one stripe is lost as a result. Inner node disjoint trees are constructed by selecting roots that vary in id prefixes. MaxDisjoint creates disjoint paths in much the same way in Pastry.

2.2 Node Id Assignment

Node id assignment has generally aimed to achieve network diversity, uniform id distribution, and security. Most well-known distributed hash table implementations [32, 39, 43] rely on random id assignment. Node ids are generated using a secure hash function applied to the IP address of the node, for instance. Since nodes with similar IP addresses are unlikely to have similar node ids, the id space is diverse in terms of the network location of neighboring nodes. This reduces the probability that a catastrophic network outage partitions the overlay. If we assume that catastrophic network outages are rare, then the benefits of random node id assignment are limited. In these situations, we propose an id assignment scheme that improves the cost-consciousness of the overlay.

A second desirable property of random id assignment is the uniform distribution of ids. A skewed node id distribution can severely affect load balance. When very few nodes are responsible for a large portion of the id space, the storage burden on those nodes is greater than nodes within a denser portion of the id space. We combat this problem using a technique similar to the approach used in [44]. Zhou et al. propose a hierarchical location-based id assignment policy to improve locality. Nodes within a common region are assigned a common prefix. To ensure load balance, id space segments (as defined by prefix lengths) are assigned in proportion to the number of nodes in that region. Our approach differs in that our id assignment is organization-based rather than locality-based; our primary goal is to reduce inter-organization communication instead of latency.

Finally, the security implications of node id assignment have been considered. The Sybil [12] and Eclipse [35] attacks are well known attacks on the id assignment policy. These attacks focus on gathering multiple ids in the overlay to partition the network or "eclipse" a group of nodes from the rest of the network. Castro et al. [7] propose using certified random node ids in which a central authority provides certificates to bind a random node id to a node. Rather than using cryptographic techniques, we employ replica placement to maintain object availability in the face of network partitions.

2.3 Replica Placement

Replica placement has long been studied in the realm of distributed computing. Specifically in the area of peer-to-peer networks, it has become clear that replica placement can be used to manage load and satisfy capacity constraints while improving the quality of service in the system. Many studies have compared the performance of several placement schemes in terms of quality of service, availability, and time to recovery [10, 13, 24, 28]. However, to our knowledge, none have considered routing robustness as it relates to replica placement.

Chen et al. [10] propose a replica placement that builds a dissemination tree to propagate updates efficiently. This placement creates good load distribution and small delay and bandwidth costs for replica updates.

Lian et al. [24] study the impact of replica placement on reliability in distributed block storage systems. They compare sequential, random, and striped placement in these systems to determine which provides the most reliability. They find that both sequential and random placement have positive and negative aspects and prescribe the striped placement to reach a near-optimal compromise.

Douceur and Wattenhofer [13] compare multiple heuristics for determining the dynamic placement of replicas in Farsite, a scalable, distributed file system [2]. Replicas

are shifted to find the placement that meets the availability constraints defined by each heuristic. Through their analysis they are able to measure the algorithmic efficiency and the efficacy of each placement. With these metrics in hand, they prescribe a placement for Farsite.

On et al. [28] use availability and uptime heuristics to determine a dynamic replica placement that provides the best quality of availability. From their study, they determine that the placement of replicas can have a greater impact on the quality of availability than the number of replicas. In other words, although increasing the replication degree may increase the hit rate, it does not necessarily improve the quality of availability. This is precisely the behavior we observe in peer-to-peer networks where the availability is influenced by the number of replicas and the ability to retrieve them over multi-hop paths.

Ghodsi et al. [16] propose the symmetric replica placement scheme, wherein replica ids are equally spaced over the entire id space. This technique reduces the message overhead in node joins and leaves, provides better load balance, and improves fault tolerance. We highlight this work because equally-spaced and symmetric replica placement are equivalent in Chord; therefore, the benefits demonstrated in [16] can be realized in our approach as well. However, it is worth noting that MaxDisjoint placement provides added benefit in other DHT implementations that equally-spaced placement cannot achieve. Furthermore, our work is the first to consider the routing robustness benefits of replica placement.

2.4 Cost-Conscious Overlays

Most overlay designs have focused on costs to the client – latency, for instance. Locality- or topology-awareness is becoming a prevalent feature in distributed hash table implementations. Most notably, Pastry [32], Tapestry [43], and CAN [31] all aim

to reduce latency by employing some form of locality-awareness. Although locality-awareness is beneficial to the clients of a DHT, it does not dramatically reduce the economic burden on network operators. Therefore, we focus on the costs to the Internet service providers hosting the lookup service, rather than its clients.

Seetharaman and Ammar [34] show that overlay routes may violate the administrative inter-domain policies that govern Internet routing. Violating these policies, like the valley-free property of Internet routing, incurs cost for network operators. Seetharaman and Ammar propose inserting nodes into the overlay to create additional peering relationships and avoid policy violations. This work differs from ours in its focus; although the elimination of policy violations may have an effect on the overall cost to ISPs, it is not goal of the approach. The approach aims simply to eliminate inter-domain policy violations, which does not necessarily reduce transit costs.

Xie et al. [42] propose a new framework for cooperative traffic control between peer-to-peer applications and network operators. The framework prescribes deploying an iTracker for each network provider, which provides applications with the network topology, provider policy, and network capability information. Although this would provide peer-to-peer applications with the necessary information to operate in a cost-conscious manner, it may be difficult in practice to persuade a provider to divulge its network policy and capability. With our approach, the hosting organizations do not need to divulge any information regarding their peering or transit policies. Instead, the aggregate cost for resolving a query is expressed for each organization using the cost function we define. A global replication degree is determined by minimizing the organization cost functions in concert or by utilizing one of several replication degree selection strategies we provide.

CHAPTER III

DISTRIBUTED HASH TABLES

Structured overlay architects borrowed a well known data structure to serve as a model for their designs: the hash table. Hash tables store key-value pairs such that a value can be retrieved efficiently given the corresponding key. Distributed hash tables (DHTs) function in much the same way; key-value pairs are stored at appropriate locations in a network such that values can be efficiently retrieved given a key. Each node manages a small number of references to other nodes to facilitate lookups, which effectively distributes routing information. The result is retrieval time and storage costs both on the order of $O(\log N)$.

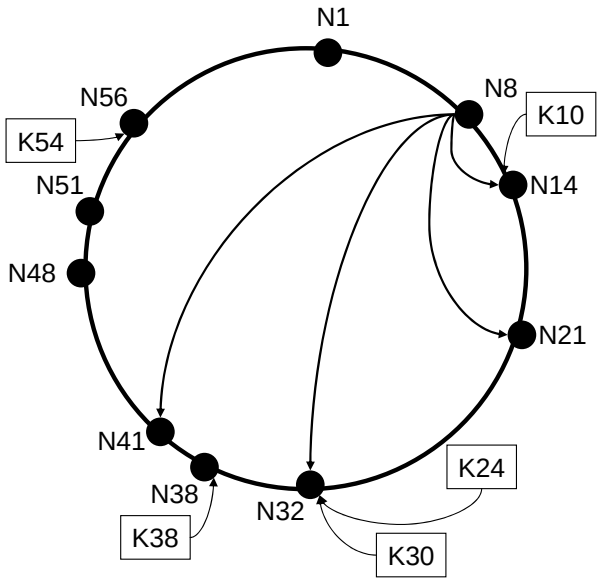
Distributed hash table implementations differ primarily in the management and partitioning of the address space. Typically, each scheme lends itself to a geometric interpretation of the address space (e.g., ring, hypercube, torus). Furthermore, each implementation specifies a routing scheme that exploits the system structure to efficiently look up values.

In this chapter, we will describe Chord and Pastry, two DHT implementations that we refer to extensively in the remainder of the dissertation.

3.1 Chord

The simplest and perhaps most elegant DHT implementation is Chord, published by Stoica et al. [39]. The address space is an m-bit ring; that is, ids form a one-dimensional id circle modulo 2^m wrapping around from $2^m - 1$ to 0. Typically, a secure hash function is used to generate ids, either by hashing an IP address for node ids or by hashing some metadata for data items.

The id of a data item is called a *key*. Formally, the $(key, value)$ pair (k, v) is



Finger Table of N8	
Target id	Successor
N8 + 1	N14
N8 + 2	N14
N8 + 4	N14
N8 + 8	N21
N8 + 16	N32
N8 + 32	N41

Figure 3: A 6-bit Chord ring. Node ids are labeled N$\langle id \rangle$ and keys are labeled K$\langle key \rangle$. The finger table and corresponding finger links are shown for node N8.

hosted by the node whose id is greater than or equal to k. This node is called the *successor* of key k. This scheme, called *consistent hashing*, implies that each node is responsible for all the keys less than or equal to its own id, but greater than the id preceding it in the ring.

An example of a Chord ring with $m = 6$ is depicted in Figure 3. The simplest method for finding a key k is to traverse the ring in the clockwise direction starting at any node. The search stops when the successor of k is reached. Although simple, this method is very inefficient. To route efficiently in a Chord ring, each node maintains a routing table called a *finger table* (see Figure 3). In a ring with an m-bit identifier space, the finger table has at most m entries. At node n, the finger table entry at row i contains the successor of $n + 2^i$, where $0 \leq i \leq m - 1$. For the example in Figure 3, the fourth finger table entry of node $N8$ contains the successor of $8 + 2^3 = 16$, or $N21$.

When routing to a key k, a node forwards the lookup to the closest predecessor of k in the ring using its finger table. When the lookup reaches a node n such that k lies between n and the successor of n, node n reports its successor as the lookup

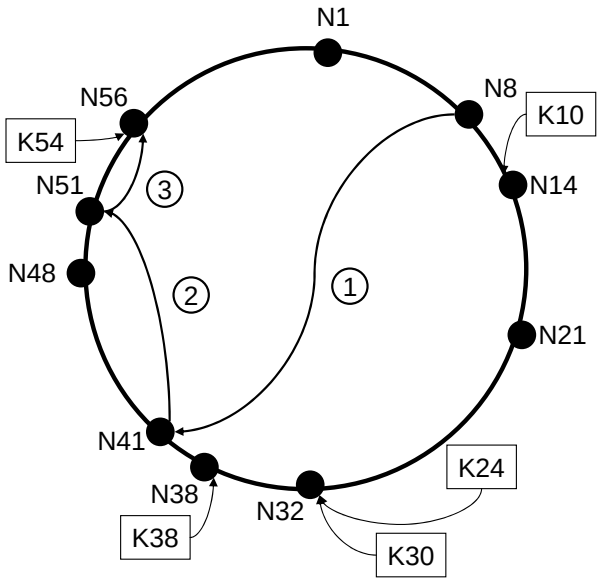


Figure 4: An example route in a 6-bit Chord ring. The route corresponds to a lookup for K54. The routing hops use the finger table entries N8 + 32, N41 + 8, and N51 + 4, respectively.

result. Intuitively, with each hop, the lookup reduces the distance between the current hop and the key by at least half. This greedy routing algorithm results in $O(\log n)$ routing hops, where n is the number of nodes. An example route is shown in Figure 4. Note that the distance traveled at each hop decreases by at least half as the message proceeds down the lookup path.

3.2 Pastry

Pastry, which was proposed by Rowstron and Druschel in 2001 [32], is similar to Chord, but attempts to minimize a network proximity constraint in addition to minimizing the number of hops in the overlay.

The address space is organized as an id ring, much like Chord. However, each id is thought of as a sequence of digits in base 2^b. Contrary to the consistent hashing approach of Chord, a key is assigned to the node with the id numerically closest to the key.

At each routing step, the message is forwarded to the node whose id shares a prefix

N10233102

Leaf Set

10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Neighborhood Set

13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

Routing Table

-0-2212102	**1**	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	**2**	10-3-23302
102-0-0230	102-1-1302	102-2-2302	**3**
1023-0-322	1023-1-000	1023-2-121	**3**
10233-0-01	**1**	10233-2-32	
0		102331-2-0	
		2	

Figure 5: Example Pastry node state. Shown for hypothetical node 10233102, $b = 2$, and $|L| = |M| = 8$. The top row of the routing table is row zero. The shaded cell in each row shows the corresponding digit of the present node's id. The node ids in each entry have been split to show the ⟨common prefix⟩- ⟨next digit⟩-⟨rest of ID⟩.

with the key that is at least one digit (or b bits) longer than the prefix shared with the current node's id. If no such node exists, the message is forwarded to a node with shared prefix of equal length to the current node, but with id numerically closer to the key. Therefore, at each routing step, the message moves to nodes numerically closer to the key. To support this routing scheme, each node maintains a routing table, neighborhood set, and leaf set, which are described in the following paragraphs.

3.2.1 Routing Table

A Pastry routing table is organized into $\lceil \log_{2^b} N \rceil$ rows, where N is the identifier space size. Each row contains $2^b - 1$ entries, where the entries at row i of the routing table refer to a node that shares a prefix of i digits with the current node, but whose $(i+1)$-th digit has one of the $2^b - 1$ possible values other than the $(i+1)$-th digit of the current node. The entry in the j-th column of row i has the value j in the $(i+1)$-th digit. In the event that no such node fits within the constraints of a given routing table entry, the entry is left empty. An example routing table is shown in Figure 5.

As you progress down the rows of the routing table, the entries cover smaller

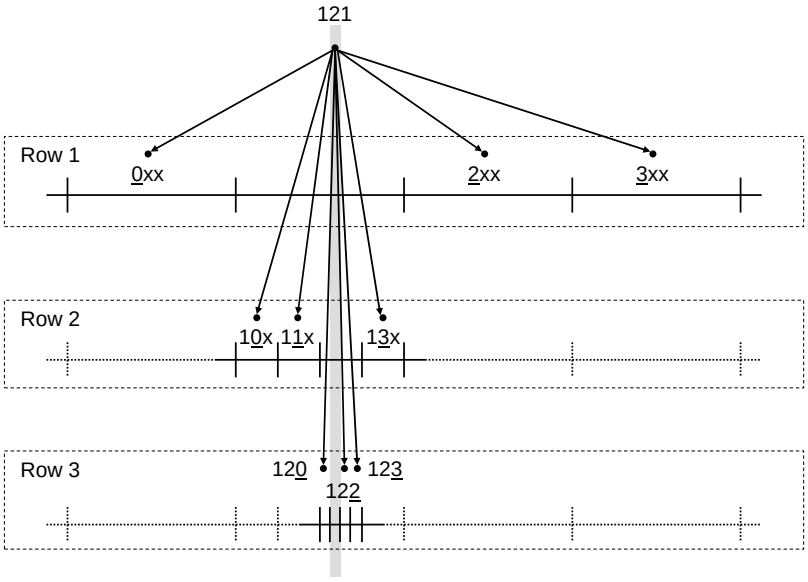


Figure 6: Distribution of Pastry routing table entries in the id space for the hypothetical node 121 with $b = 2$ and $N = 2^{3b}$. The highlighted region denotes the segment within which the node 121 lies.

segments of the id space. As a result, each node maintains fine-grained routing information about the segments nearest it in the id space and coarse-grained information about those segments far away. We depict this graphically in Figure 6.

Note that there could be multiple candidates to occupy a particular entry in the routing table. Rather than randomly selecting one of the candidates, the node that is selected is the "closest" to the present node, in terms of some network proximity metric. By building the routing table in this fashion, Pastry is able to minimize a physical network locality constraint in addition to the number of routing hops in the overlay.

3.2.2 Neighborhood Set

To facilitate the construction of the routing table, specifically to find the closest nodes according to the proximity metric, each node maintains its neighborhood set M, which contains its closest neighboring nodes. This set is useful for joining nodes that need to construct their routing table and also for maintaining the routing table

in the event of node joins, leaves, or failures. A sample neighborhood set is shown in Figure 5. Note that the ids of the nodes in node n's neighborhood set do not necessarily share any common prefix with n; instead, they are "close" to n in the underlying network.

3.2.3 Leaf Set

To aid when an appropriate routing table entry cannot be found, each node maintains a leaf set L that consists of the $\frac{|L|}{2}$ nodes with numerically closest ids larger than the current node's id and the $\frac{|L|}{2}$ nodes with numerically closest smaller ids. The leaf set and neighborhood set are usually equal in size and have typical sizes of 2^b or 2×2^b. A sample leaf set is shown in Figure 5. Note that the ids of the nodes in node n's leaf set do share a prefix with n, but they are not necessarily "close" to n in the underlying network.

3.2.4 Routing

When routing a message, the node first determines if the key falls within the range of ids covered by its leaf set. If so, the message can be forwarded directly to the destination node.

If the key is not in the leaf set, then the routing table is used. The length l of the shared prefix between the current node's id and the key is computed. If there exists a non-empty entry that shares $l+1$ or more digits with the key, the message is forwarded using that entry. If the entry is empty, the message is forwarded to a node in its leaf set, neighborhood set, or routing table with shared prefix of at least l that is numerically closer to the key than the current node.

This routing procedure always converges because each step takes the message to a node that either (1) shares a longer prefix with the key than the current node, or (2) shares the same prefix with the key as the current node, but is numerically closer to the key. It can be shown that the expected number of routing hops is $O(\log n)$,

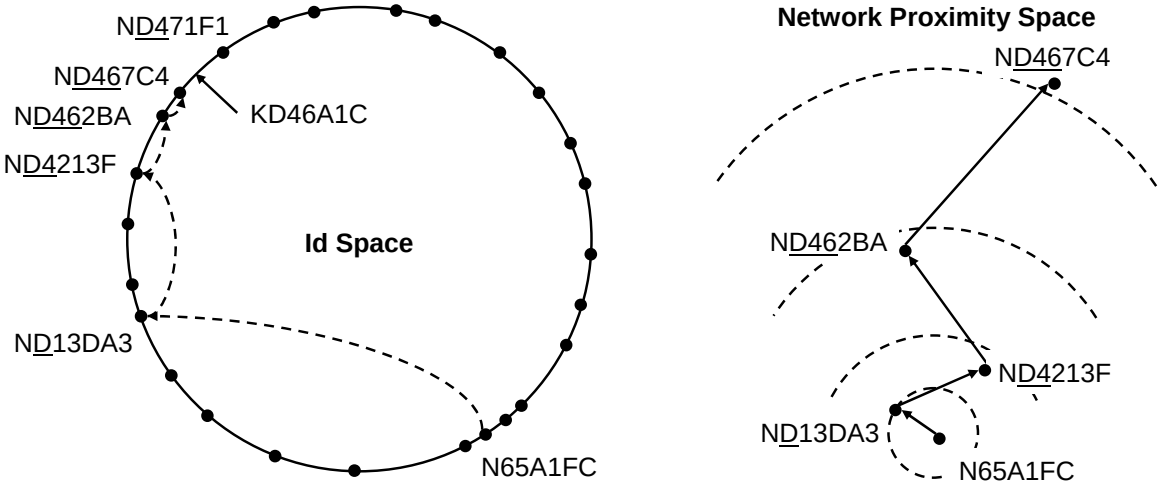


Figure 7: The relationship between id space proximity and network proximity in Pastry routes. The route from N65A15C to KD46A1C ($b = 4$) is shown in the id space (left) and the network proximity space (right). Node ids are shown with the matched prefix underlined.

where n is the number of nodes in the DHT.

As mentioned, Pastry routing attempts to minimize the routing hops in the overlay while minimizing network proximity by selecting routing table entries that are "closest" in terms of the proximity metric. With these somewhat conflicting goals, Pastry routes exhibit the behavior depicted in Figure 7. The first hops of a Pastry route are very efficient; a long distance is traveled in the id space without moving far from the source in the proximity space. However, as the route approaches the destination, this efficiency is lost. The necessity for convergence in the id space hinders the effectiveness of Pastry's locality awareness and hops become longer in the network proximity space.

3.3 Distributed Hash Tables with Tree-Based Routing

Distributed hash tables are often referenced by their geometry, i.e., ring (Chord [39]), torus (CAN [31]), tree (Plaxton [29]), or some hybrid (Pastry [32], Tapestry [43]). This geometry impacts neighbor and route selection, which, in turn, has an impact on flexibility, resilience, and proximity performance as studied in [17]. Although we

use the term "tree-based routing", we are not referring to the geometry, but the routing algorithm. Tree-based routing algorithms have specific properties that will be defined herein, but we begin with an example of the commonalities between Chord and Pastry routing, two tree-based routing algorithms, to provide the reader some intuition.

3.3.1 Routing Similarities of Chord and Pastry

When the parameter $b = 1$, Pastry is very similar to Chord. To reveal the similarity in their routing tables, Pastry routing table entries must be constrained in a manner similar to what is described in [7]. The constraint that we apply supersedes the physical network locality constraint that is used to choose routing table entries. Instead, we place a constraint on the *suffix* of each routing table entry. The suffix of an entry in the i-th row of the routing table consists of the $L - (i + 1)$ least significant digits, where $L = \log_{2^b} N$ and N is the id space size. The constraint requires that the suffix of each entry match the suffix of the current node.

Since Chord fingers are constructed by adding powers of two rather than manipulating id prefixes as in Pastry, the entries in the finger table of node n will not necessarily have a prefix identical to n's id. If the $(L-i)$-th bit of n's id is zero where L is the id length, then i-th finger will be similar to an entry in the Pastry routing table. However, if the $(L - i)$-th bit is 1, the prefix changes to reflect the carryover of adding. These differences are shown graphically in Figure 8. Note that the right columns of each table are identical; these entries correspond to the bits in $N37$'s id that have the value zero.

The similarities in routing table entries results in a similar routing algorithm. To be precise, we claim that Chord and Pastry both use a "tree-based" routing scheme. At each hop in the route, the routing algorithm finds the routing table entry responsible for the contiguous segment of the id space in which the key lies. The message

N100101 (N37)	
N000101 (N5)	**1**
0	N110101 (N53)
0	N101101 (N45)
N101001 (N41)	**1**
0	N100111 (N39)
N100110 (N38)	**1**

Chord Finger Table

N100101 (N37)	
N000101 (N5)	**1**
0	N110101 (N53)
0	N101101 (N45)
N100001 (N33)	**1**
0	N100111 (N39)
N100100 (N36)	**1**

Constrained Pastry Routing Table (b = 1)

Figure 8: Comparison of Chord finger tables and Pastry routing tables. Ids are shown in binary and decimal (in parenthesis). The Chord finger table has been reorganized for ease of comparison. The Pastry routing table entries are constrained such that each entry's suffix matches that of the current node. Entries that differ are shown in *italics*.

is forwarded to that entry and the process is repeated until the key is reached. If we aggregate the routes from any source node to every possible destination, the resulting topology is a tree. We show the id space segmentation for both Chord and Pastry in Figure 9. Although Chord has a ring-based geometry and Pastry has a hybrid ring- and tree-based geometry, both DHTs have a tree-based routing algorithm. We will discuss tree-based routing in detail and formally prove its properties in Chapter 5.

The subtle difference between Chord and Pastry routing is in the routing table construction, which leads to the differing geometries. Both keep several routing table entries to route to ids nearby and few entries to route to ids far away in the id space. However, Chord nodes maintain fine-grained routing information about those nodes closest to it in the id space in the clockwise (increasing id) direction. Therefore, as we iterate down the finger table, each finger table entry is responsible for a larger segment of the id space.

To the contrary, Pastry is able to route in either direction in the id space and each node maintains fine-grained routing information about those nodes with which it shares the longest prefix. As we iterate down a Pastry routing table, the shared

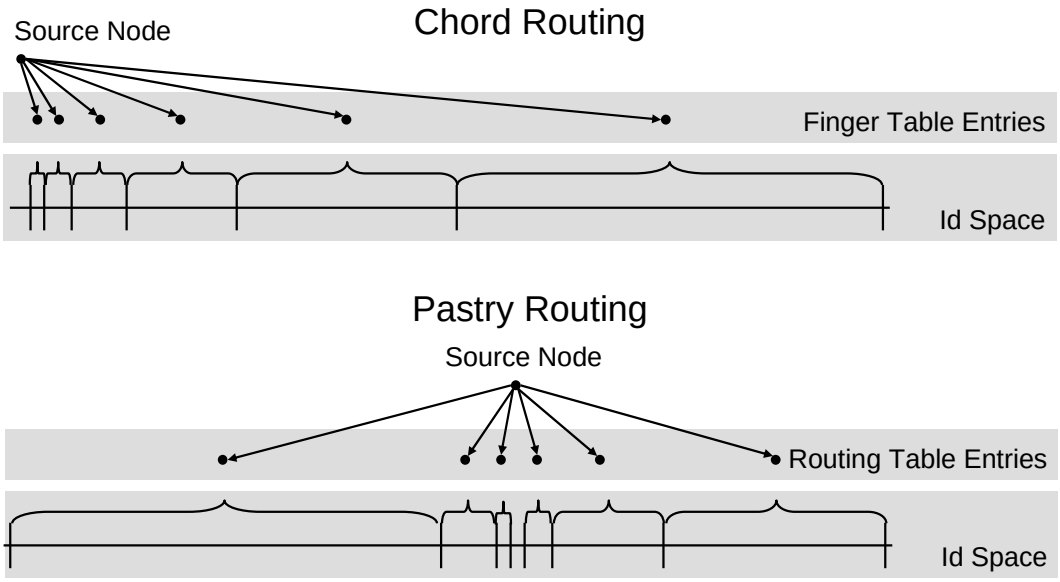


Figure 9: Comparison of tree-based routing in Chord (top) and Pastry (bottom).

prefix length increases and each entry is responsible for a smaller segment of the id space. DHTs with prefix-matching schemes [29, 32, 43] exhibit similar behavior.

CHAPTER IV

REPLICA PLACEMENT FOR ROUTING ROBUSTNESS IN CHORD

Structured peer-to-peer networks provide the benefits of efficiency, scalability, resilience, and self-organization when building distributed applications. These systems achieve efficiency by utilizing their structure to reduce redundant operations. The structure brings added responsibility for each node participating in the system. For instance, Chord [39] and other distributed hash table (DHT) implementations depend on each node to efficiently route lookups. Therefore, each node has the power to route and *misroute* lookups. Malicious routing is not the only vulnerability; adversaries may act upon several attack vectors. Before DHTs can be applied as real world solutions to problems where dependability is paramount, like the domain name system and voice over IP systems, these vulnerabilities must be addressed.

In this chapter, we consider two threats to DHTs: malicious routing and storage and retrieval attacks. In a storage and retrieval attack, an adversary may seize control of a key or a set of keys by selecting an appropriate node id. Then, the adversary may deny access to those keys. Replication is commonly used to mitigate this threat while multipath routing addresses routing attacks. Our solution integrates these two approaches into a replica placement scheme for Chord that can be tuned to produce a desired number of disjoint routes.

4.1 Background

Sit and Morris [36] identify the major threats to structured peer-to-peer systems to be:

1. Routing attacks, in which a malicious node may misroute lookups or attempt to corrupt routing tables;

2. Storage and retrieval attacks, in which a malicious node may claim that a key it serves does not exist; and

3. Miscellaneous attacks, in which a malicious node may act arbitrarily to tamper with the correct operation of the system. These attacks include Sybil and Eclipse attacks [12, 35] as well as denial of service (DoS) attacks [14, 38].

Our solution mitigates the first two threat types. As suggested in [37], routing attacks can be mitigated with alternative lookup paths. To this end, they propose the notion of *independent lookup paths*. Two routes are said to be independent if they share no common node other than the source and destination nodes. Independent routes help improve routing robustness, but do not address storage and retrieval attacks because the routes share a common destination node. An adversary can compromise the destination and all the multipath effort is for naught. Instead, we suggest an approach that creates *disjoint routes*, which we define to be routes that share no common node other than the source. This is a more robust notion of alternative paths.

We build our solution as a replica placement scheme because replication is very effective in mitigating the effects of storage and retrieval [36]. The novelty stems from the intuition that the placement of replicas can help defend against routing attacks by creating route diversity. Furthermore, a replica placement can be deployed as a middleware without significant modification to the underlying peer-to-peer system.

For disjoint routes to improve the robustness of our system, we require that the keys stored in the system be self-verifying; that is, we assume that a key's integrity can be verified by the client. This is necessary to detect when a node returns a corrupted object. For types of data where self-verification is not natural, we can generate key

ids by hashing the object as in [11]. The client can verify the object's integrity by comparing its hash with the key id. Castro et al. [7] discuss self-verifying data as well as a method for managing mutable self-verifying data.

Under these assumptions, our system can tolerate attacks by up to $d-1$ malicious nodes acting arbitrarily, where d is the number of disjoint routes. We also prove, under the same assumptions, that the approach can tolerate a *run* of contiguous nodes controlled by an adversary covering more than one-quarter of the identifier space. To our knowledge, this is the first approach to peer-to-peer routing for which properties such as these have been formally proven.

4.2 Equally-Spaced Replication

In this section, we discuss a replica placement scheme that produces disjoint routes for each replica set. The simplest method for generating disjoint routes is to ensure that the routes are contained within non-overlapping segments of the ring; this is the premise of our work.

As successor list (neighbor set) replication is implemented in Chord [39], keys are replicated in a chain of nodes starting at the key's home node. To access a correct replica, the lookup begins at the first node in the chain and iterates down the chain until a correct replica is found. It is clear that the routes to each of the replicas differ only in the final hop. An adversary could very easily deny access to the entire replica set by controlling a single node in the overlapping portion of the routes.

In our approach, we place replicas at equally-spaced locations over the entire Chord id space. Rather than using the same route for each replica, we initiate separate lookups for each object in the replica set. Our placement scheme generates disjoint routes, which improves routing robustness.

4.2.1 Evaluation of Disjoint Routes

The proposed solution is simple: replicas are assigned equally spaced ids starting at the master key id and proceeding in a wrap-around fashion over the entire id space. For example, consider a Chord ring of size 256 with a replication degree of 4. The four replicas of key 71 will be assigned ids 71, 135, 199, and 7. Note that the replicas are equally spaced; that is, the difference between consecutive replica ids is equal to the total id space size divided by the replication degree (e.g., $135 - 71 = 64 = \frac{256}{4}$). We claim that this replication scheme produces a predictable number of disjoint routes. To prove this claim, we first show that routes originating from a common query node that differ in the first hop must be disjoint.

Lemma 1. *Chord routes originating from a common query node that differ in the first hop are disjoint.*

Proof. Consider a query node q and two routes originating at q destined for keys k_1 and k_2[1]. Suppose that the first hops of the routes to these keys are $q + 2^i$ and $q + 2^j$, respectively, where $i \neq j$. According to the definition of the Chord protocol, $k_1 \in [q + 2^i, q + 2^{i+1})$ and $k_2 \in [q + 2^j, q + 2^{j+1})$. Furthermore, each hop of these routes must reside in their respective intervals. Therefore, the routes from q to k_1 and k_2 are disjoint. □

This lemma, though straightforward, provides insight into the premise behind our solution. To produce disjoint routes from a common source, the first hops in each route must be different. The first hop in a route is determined by the distance between the source and destination nodes. Therefore, if we place replicas at carefully chosen locations, we can produce disjoint routes.

We prove our claim within the context of a *full Chord ring*. We define a full Chord ring to be a Chord ring wherein all possible ids are represented. In other words, for

[1] All ids are ordered on an *id circle* modulo n.

any key k, k will be located at the node with id k.

Theorem 1. *To produce d disjoint routes to a key k from any query node in a full Chord ring of size n with equally-spaced replicas ($d \leq \log_2 n + 1$), k must be replicated at exactly 2^{d-1} locations in a wrap-around fashion starting at k.*

Proof. (by induction) In a full Chord ring of size n with $d = 2$, k is replicated at nodes k and $k + \frac{n}{2}$. Consider a query node q. Let k_1 be the replica in the interval $\left[q, q + \frac{n}{2}\right)$ and k_2 be in the interval $\left[q + \frac{n}{2}, q\right)$. (Since k_1 and k_2 are equally spaced in the ring, they must reside on different halves of the ring.) The first hop of the route from q to k_1 must be in $\left[q, q + \frac{n}{2}\right)$ and the first hop of the route from q to k_2 must be $q + \frac{n}{2}$. Applying Lemma 1, the routes from q to k_1 and k_2 are disjoint.

Assume 2^{d-1} equally-spaced replicas produce d disjoint routes to k. Let I be the largest interval $[q, q + 2^i)$ that contains exactly one replica of k. Since the replicas are equally spaced and $d < \log_2 n + 1$ (i.e. there are fewer replicas than nodes in the system), the interval I exists. Increasing the replication degree to 2^d doubles the replica density in the id space and guarantees that there are two replicas in I. Since the 2^{d-1} replicas that formed d disjoint routes are a subset of the new set of 2^d replicas, it is enough to show that the two replicas in I produce disjoint routes. Let k_1 and k_2 be the ids of these two replicas such that $k_1 \in [q, q + 2^{i-1})$ and $k_2 \in [q + 2^{i-1}, q + 2^i)$. The first hop to k_1 is in $[q, q + 2^{i-1})$ and the first hop to k_2 is $q + 2^{i-1}$. Applying Lemma 1, the routes to k_1 and k_2 are disjoint. □

Although these claims are proven for a full Chord ring, we will demonstrate, through extensive simulation, that these concepts can be applied to sparsely populated Chord rings with similar performance.

4.2.2 Toleration of Runs

In this section, we introduce the concept of a *run*, the way in which a run can be used to disrupt the system, and the degree to which equally-spaced replication can

tolerate runs. We define a run of length l to be a contiguous set of l nodes from the Chord ring. Equivalently, the run of length l starting at node m is denoted in set notation by $[m, m + l)$.

As indicated in [39], an adversary can create imbalance in the distribution of nodes in the ring by appropriately selecting ids. In the worst case, an adversary can take control of a contiguous sequence of ids or, using our terminology, a run of nodes.

We will show that equally-spaced replication can tolerate adversarial runs with bounded length. Before proving the tolerable length of a run, we provide some intuition of how a run may be used to disrupt routing in the ring. Consider a query node q. An adversary can reduce the number of replicas reachable from q by half by controlling the node $q + \frac{n}{2}$. This is because all of the replicas in interval $[q + \frac{n}{2}, q)$ are routed through the node $q + \frac{n}{2}$ (half of the replicas are in this interval). If the adversary controls a run of nodes ending at $q + \frac{n}{2}$, he can control a larger number of replicas.

Theorem 2. *A full Chord ring of size n with 2^{d-1} ($d > 2$) equally-spaced replicas can tolerate any adversarial run of length $1 + n(\frac{1}{2} - \frac{1}{2^{d-1}})$.*

Proof. (by induction) Consider a query node q. For $d = 3$, there exists a replica $k \in \left[q, q + \frac{n}{4}\right)$. (There are four replicas; one must reside in the first quadrant of the ring starting at q.) If we assume that the adversary has control of the node $q + \frac{n}{2}$ (which is the worst case), then we must ensure that k is not in the run. Thus, the maximum length run is $[q + \frac{n}{4}, q + \frac{n}{2} + 1)$, which has length $\frac{n}{4} + 1$ or $1 + n(\frac{1}{2} - \frac{1}{2^2})$.

Assume that the longest tolerable run in a full Chord ring with 2^{d-1} equally-spaced replicas has length $1 + n(\frac{1}{2} - \frac{1}{2^{d-1}})$. Consider a query node q. If we assume that the adversary takes control of the node $q + \frac{n}{2}$, then he does not control any nodes in the interval $[q, q + \frac{n}{2^{d-1}})$. There must exist at least one replica k in this interval. If we double the replication degree to 2^d, then there are two replicas in this interval separated by $\frac{n}{2^d}$. Thus, the length of the tolerable run increases by $\frac{n}{2^d}$ to

$1 + n(\frac{1}{2} - \frac{1}{2^d})$. □

Note that with four equally spaced replicas, an adversary may control of a run of more than a quarter of the nodes in the Chord ring and there will be a route to one replica for every possible query. As the replication degree approaches the id space size, the maximum tolerable length approaches one-half the id space.

4.2.3 Replication Degree Flexibility

Thus far, our discussion has assumed a fixed replication degree for every key inserted into the ring. However, it is oftentimes the case that certain objects inserted into the system are more critical than others. It would be desirable to replicate these objects to a higher degree and maintain the benefits of equally-spaced replication.

The assumption of a fixed system-wide replication degree is helpful because it allows any node to compute the locations of all replicas for a given key. However, this is not a necessary condition. For example, suppose we have a Chord ring wherein the most critical objects should be stored at 16 equally-spaced locations throughout the ring. We can replicate objects that are less critical at 8, 4, or 2 locations in the ring. Since the replicas are equally-spaced, we can guarantee that the actual replica locations will be a subset of the 16 possible locations regardless of the key's actual replication degree. The side effect of this approach is that some replica requests may result in a "object not found" response. However, this added cost comes in lieu of the much larger cost of maintaining the replication degrees of all objects at each node.

4.2.4 Implementation

To uniquely identify each replica, we use a *key id pair* (k, v), where k is the *key id* and v is *virtual key id*. For each replica, v gives the location of the master key. By definition, the master key k is denoted by the pair (k, k).

When a key k is inserted into a Chord ring of size n with replication degree d, we first compute the key id pairs for each replica: $(k, k), (k + \frac{n}{d}, k), (k + \frac{2n}{d}, k), \ldots (k +$

$\frac{(d-1)n}{d}, k)$. For example, consider a Chord ring of size 256 with replication degree 4. When the key 71 is inserted, we compute $(71, 71)$, $(135, 71)$, $(199, 71)$, and $(7, 71)$.

Once the key id pair for each replica is computed, we use the traditional Chord key insertion mechanism to insert the replicas. That is, we perform a lookup for each key id and store the replica at those locations. In the example above, we lookup keys 71, 135, 199, and 7 and store the replicas at their respective locations.

Next, the Chord lookup primitive must be modified to accommodate the new replication scheme. When a node is queried for a key, the query node must first compute the locations of all replicas (using the known ring size and replication degree). The key id is used to route to the replicas. Once a replica's home node is found, the key id pairs are compared to return the appropriate replica.

It is worth noting that no additional finger table entries are required to route to the replicas. In addition, if the query node dispatches the lookups for the entire replica set simultaneously, there may be an improvement in performance because the query node can use the first correct response received (which may have returned along a route shorter than the route to the master key). However, if the added load of the extra lookups puts strain on the system, the performance may improve only slightly or even degrade.

Finally, when nodes join or leave the ring, traditional Chord node join and leave mechanisms can be used by simply ignoring the virtual peer ids in each key id pair. Note that the traditional Chord replica placement requires modification to the node join and leave mechanisms. To maintain the replication degree, replicas will need to be shifted for every node join or leave.

4.3 Experiments

To confirm our theoretical results, simulations were performed to measure the average number of disjoint routes per replica set. To evaluate the performance of our

Table 1: Chord simulation parameters.

Symbol	Parameter
N	Identifier Space Size
n	Number of Nodes
C	Number of Clusters
σ	Cluster Width (standard deviation)
r	Replication Degree

replication scheme in the presence of an adversary, the probability of routing success per query was measured. In the following subsections, we outline the experimental setup and discuss the results.

4.3.1 Experimental Setup

A simulator was designed to model Chord rings at message-level detail; that is, the simulator computes routes from any query node to any key using finger tables. The simulator was used to measure the performance of replication schemes over uniform and clustered node distributions. To simulate a system with little or no malicious activity, a uniform distribution was used to assign node ids.

One potential attack on equally-spaced replication is to cluster the node ids, leaving unpopulated gaps in the id space. With consistent hashing and this distribution of ids, it is possible that two or more replicas may be managed by a single node, which eliminates one or more possible disjoint routes. We model clustering by selecting an id for the cluster mean and randomly sampling node ids from a Gaussian distribution centered at the cluster mean. In addition to varying the number of clusters, we can vary the size of overlap by tuning the variances of the distributions. The simulation parameters are summarized in Table 1.

Finally, we compare the performance of equally-spaced replication to other placement schemes. Namely, we consider the random placement of replicas, a variant of the replication scheme used in Chord [39], and a simple spaced replication scheme. We implement the same placement used in Chord; that is, we place replicas at the

chain of nodes starting at the key's home node. However, rather than using a single lookup to reach all replicas, we perform independent lookups for each replica, which may slightly increase the number of disjoint routes.

For the spaced replication scheme, we introduce a parameter s that can be used to indicate the spacing between replica ids in the chain. Replicas of the key k are placed the locations $k, k+s, k+2s, ..k+(r-1)s$, where r is the replication degree. Unlike the Chord replication variant, spaced replication may result in replicas being collocated for small spacings. Note that when $s = \frac{N}{r}$, this scheme is equivalent to the equally-spaced scheme.

To manage our variant of Chord replication and the random placement scheme, we maintain a mapping of each key to its replica locations. More sophisticated techniques are necessary to efficiently manage the mapping. For example, we may use multiple hash functions to generate a pseudo-random placement. Chord avoids this complexity by using a single route for all replicas. Equally-spaced replication is much simpler; replica locations can be computed directly from the key identifier.

In each experiment, we consider 10 random node distributions. For each distribution, 25 keys are selected at random and routes are computed from every node in the ring to each replica set. Each data point represents the average over a total of the 250 trials.

4.3.2 Measurement of Disjoint Routes

The first series of experiments conducted measures the performance of each replication scheme with a uniformly populated Chord ring. A Chord ring with $N = 8192$ was modeled with a 6.25% load (or $n = 512$). The average number of disjoint routes per key is shown in Figure 10. The 95% confidence intervals were computed to be less than 0.01 routes for the experiments conducted. Therefore, these results are representative of the relative performance of the schemes tested.

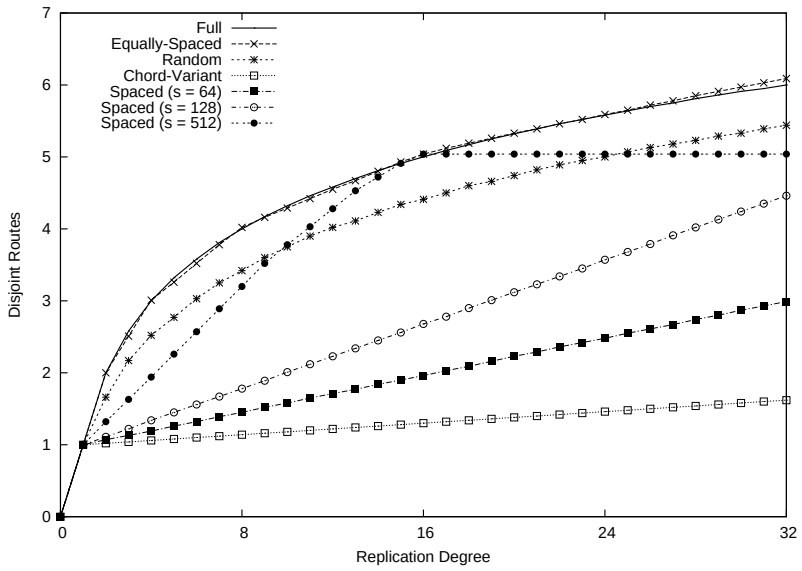


Figure 10: Disjoint routes for uniformly distributed nodes in Chord ($N = 8192, n = 512$).

The series marked "Full" reflects the number of disjoint routes expected for a full Chord ring (d for 2^{d-1} replicas). It is clear that equally-spaced replication approximates this curve and outperforms all other replication schemes. Of the replication schemes tested, the Chord replication variant performs worst in terms of the average number of disjoint routes. Unlike other placement schemes, increasing the replication degree does not significantly affect the likelihood of route overlap.

The remaining series show the increase in performance in using spaced replication with spacings of 64, 128, and 512 between replica ids. The number of disjoint routes increases linearly with the number of replicas up to $\frac{N}{s}$ replicas, where s is the spacing. At this point, the optimal number of replicas (as determined by equally spacing) is reached and no additional disjoint routes can be generated. Once the replication degree surpasses this optimal value, the replicas begin to wrap-around and overlap so that no new disjoint routes are created. This implies that to achieve the best performance, the spacing between replicas should vary with the replication degree, which is precisely what the equally-spaced replication scheme does.

A random replication scheme performs well compared to other methods and is

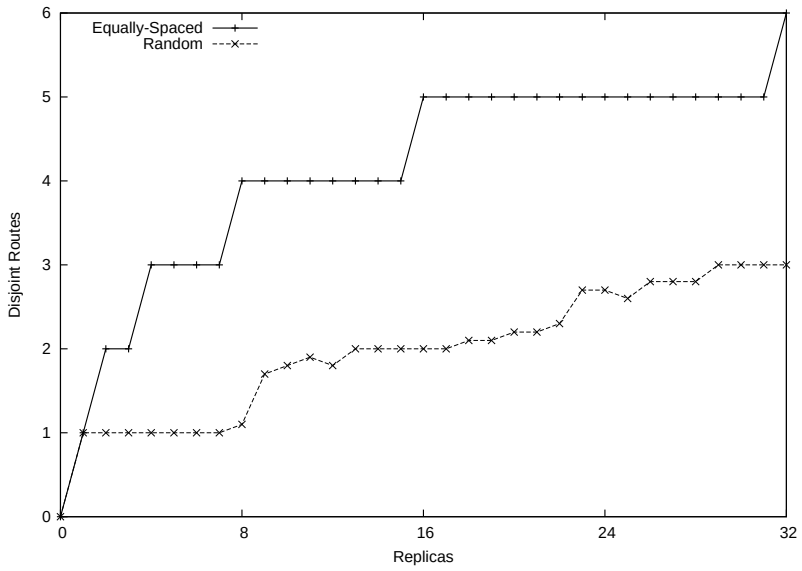


Figure 11: Worst-case performance of equally-spaced and random replica placement. The minimum number of disjoint routes created over all lookups is shown ($N = 8192, n = 512$).

only about 10% worse than equally-spaced replication. However, equally-spaced and random replication do not share comparable performance in the worst case. We measured the minimum number of disjoint routes created for every replica set in our experiments. These results are shown in Figure 11.

In the worst case, equally-spaced replica placement creates no fewer than one route less than the theoretically expected value. To the contrary, random replica placement creates 40-70% fewer disjoint routes in the worst case. Keys for which the random replica placement is poor are vulnerable to attack. To ensure that this behavior was not an isolated occurrence, we studied the distribution of disjoint routes created for the two placements. The cumulative distribution of disjoint routes created for the equally-spaced and random placement of 16 replicas are shown in Figure 12.

We expect 5 disjoint routes with 16 equally-spaced replicas. With equally-spaced placement, every replica set created at least 5 disjoint routes. In the case of random placement, 55% of replica sets produced 4 or fewer disjoint routes. In other words, the level of dependability is significantly lower than average for over half of the objects

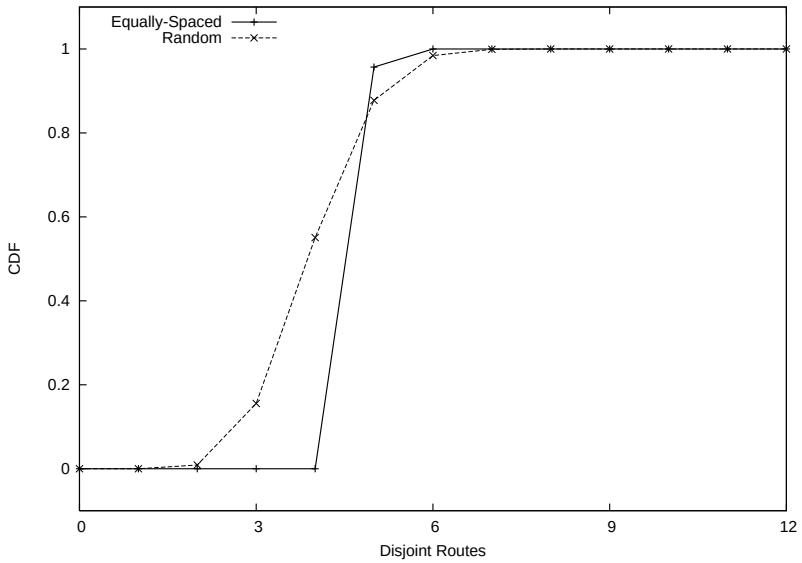


Figure 12: Cumulative distribution of disjoint routes created for the equally-spaced and random placements ($N = 8192, n = 512, r = 16$).

in the system. Equally-spaced replica placement creates disjoint routes with less variability, which provides consistent dependability across all objects.

To measure equally-spaced replication performance in sparsely populated Chord rings, N was fixed and n was varied. The results are depicted in Figure 13. For these experiments, N was fixed at 1048576 (a 20-bit id space). n is shown on a logarithmic scale to better reflect the rate of convergence. Equally-spaced replication converges to the expected full Chord ring results for $n > 1000$ or about a 0.1% load. These results scale well with network size; to achieve a near-theoretical number of disjoint routes for a 32-bit id space, the required load is less than 0.1%. Clearly, equally-spaced replication can achieve near-ideal performance for very sparsely populated Chord rings.

There does seem to be one anomaly in the convergence of the number of disjoint routes; that is, the number of disjoint routes increases above the theoretical value before converging. This is because a small fraction of the replica sets produces one additional disjoint route. This additional disjoint route comes from the replica which precedes the query node in the ring. Typically, when the ring is moderately populated,

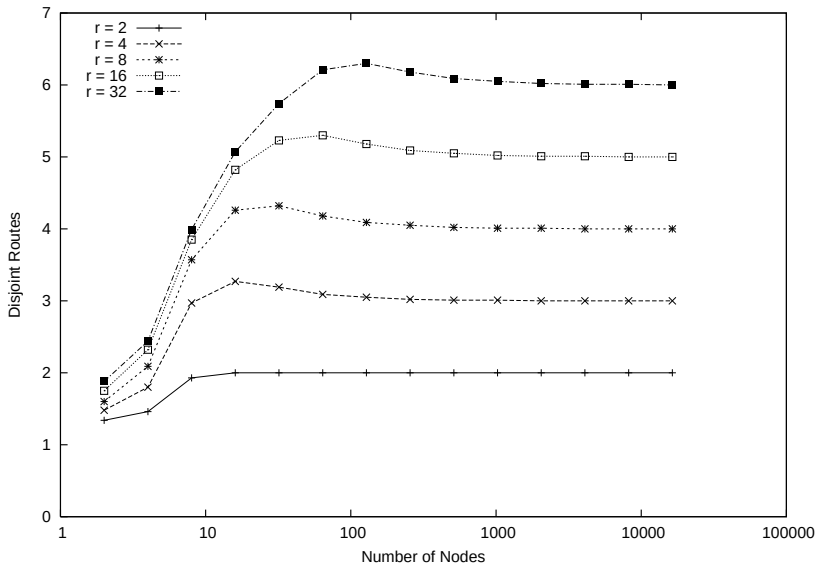


Figure 13: Disjoint routes with increasing uniform load in Chord for $r = 2$, 4, 8, 16, and 32 ($N = 2^{20}$).

this replica will be managed by a node that precedes the query node. However, in a sparsely populated ring, this replica may be managed by the query node itself. We count this zero-hop route as an additional route although no routing is actually performed. However, clearly, this replica can be used to satisfy the lookup and should be considered to improve routing robustness.

4.3.3 Resilience to Node Clustering

To model the population distribution that may result from the collusion of several malicious nodes, a series of experiments were run on clustered rings. To model a clustered distribution, four node ids were randomly selected as cluster means such that the clusters are non-overlapping and unpopulated gaps exist in the id space. The cluster density was tuned using the standard deviation of the the Gaussian distribution centered around each cluster mean. The number of disjoint routes created for $\sigma = 100$ and $\sigma = 200$ are shown in Figures 14 and 15, respectively.

Clustering can marginally reduce the number of disjoint routes created on average. The impact of clustering is intensified with tighter clusters (i.e., decreasing σ) because

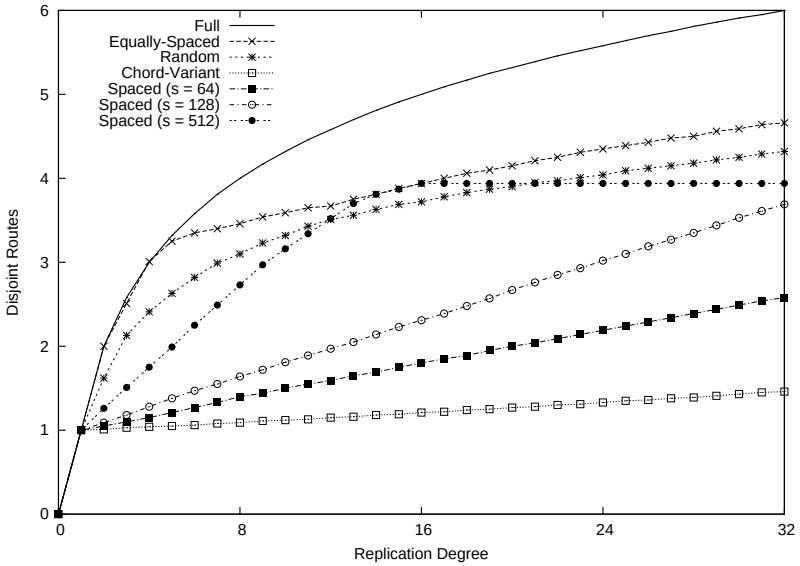


Figure 14: Disjoint routes for tightly clustered nodes in Chord ($N = 8192, n = 512, \sigma = 100$).

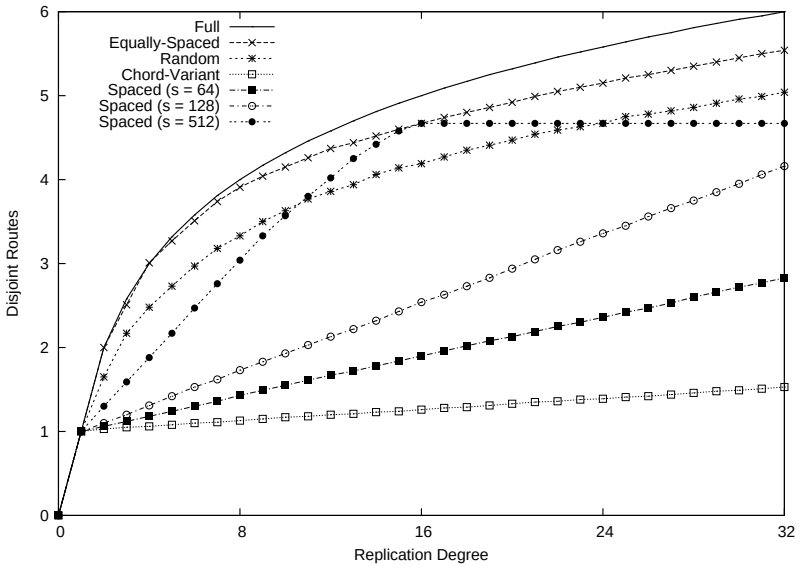


Figure 15: Disjoint routes for loosely clustered nodes in Chord ($N = 8192, n = 512, \sigma = 200$).

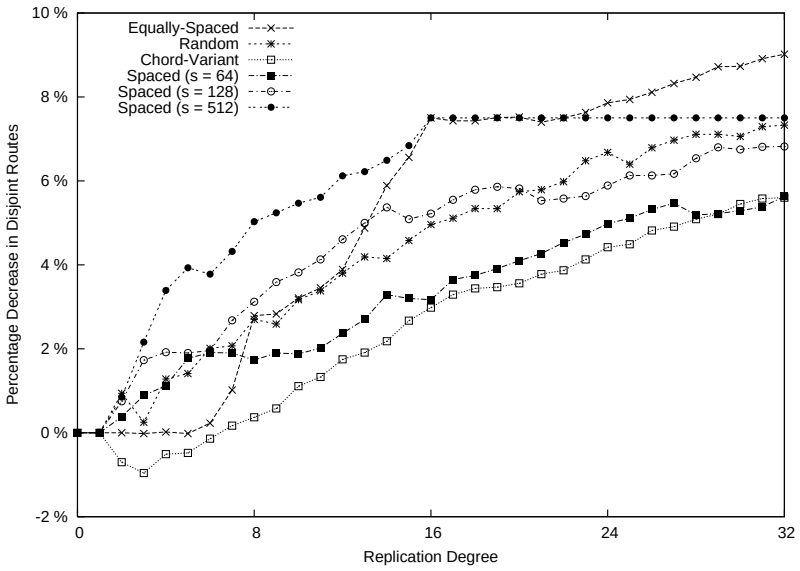


Figure 16: Percent decrease in disjoint routes from uniformly distributed nodes to a clustered distribution in Chord ($N = 8192, n = 512, C = 4, \sigma = 200$).

replicas are not placed at equally-spaced locations in the ring. Although the replica ids are equally spaced, the replicas themselves are confined to the clusters. One or more of the equally-spaced ids may lay in the unpopulated gaps between clusters. These replicas get "pushed" to next cluster in the ring, possibly eliminating a disjoint route.

Nevertheless, clustering does not affect the relative performance of the schemes tested. To determine which schemes are most resistant to clustering, we computed the percentage decrease in the number of disjoint routes from uniformly distributed nodes to a loosely clustered ($\sigma = 200$) distribution (Figure 16).

For a small number of replicas, equally-spaced replication is resistant to the effects of clustering. To have a noticeable effect, two or more replicas must fall into an unpopulated gap in the id space so they are collocated. In the case of equally-spaced replication, the inter-replica spacing must be smaller than the largest gap size. In Figure 16, there is a significant deviation for equally-spaced replication when the replication degree reaches eight. For eight replicas, the inter-replica spacing is 1024 for $N = 8192$. Using two standard deviations to capture 95% of the nodes within

each cluster, we can estimate the gap size to about 1200 for $\sigma = 200$, which is greater than the spacing. This correlates well with the data; the error does not significantly increase until a replication degree of eight is reached. Therefore, for a small number of replicas, where the inter-replica spacing is large compared to the gap size, equally-spaced replication can resist the clustering of nodes in the id space.

When the number of replicas increase, equally-spaced replication is less resistant to clustering because more replicas fall into unpopulated gaps in the ring. However, at higher replication degrees, equally-spaced replication produces a relatively large number of disjoint routes and the 7-9% reduction in the number of routes can be tolerated.

Of the schemes tested, the Chord variant seems to react the best to clustering. In fact, there is a slight improvement in its performance when the number of replicas is small. In this case, the chain of replicas may span an unpopulated gap in the id space and replicas will be located in two clusters, which will likely produce an additional disjoint route. The performance of a few replicas spanning an unpopulated gap in the id space is similar to that of a chain of many replicas. This is why the benefit decreases and disappears as the replication degree increases.

4.3.4 Impact of Replica Placement on Routing Robustness

As discussed, an adversary may use a run of nodes to create imbalance or disrupt the system otherwise. We measure the probability of routing success per query by determining the proportion of nodes for which a correct replica can be found along a route that contains only uncompromised nodes. In other words, if a route traverses a compromised node, we assume the route fails. The results are shown in Figure 17. Clearly, equally-spaced and random replication outperform the other replica placement schemes tested. As the theory indicates, equally-spaced replication tolerates compromised runs of length less than one-half of the id space well, even with a small

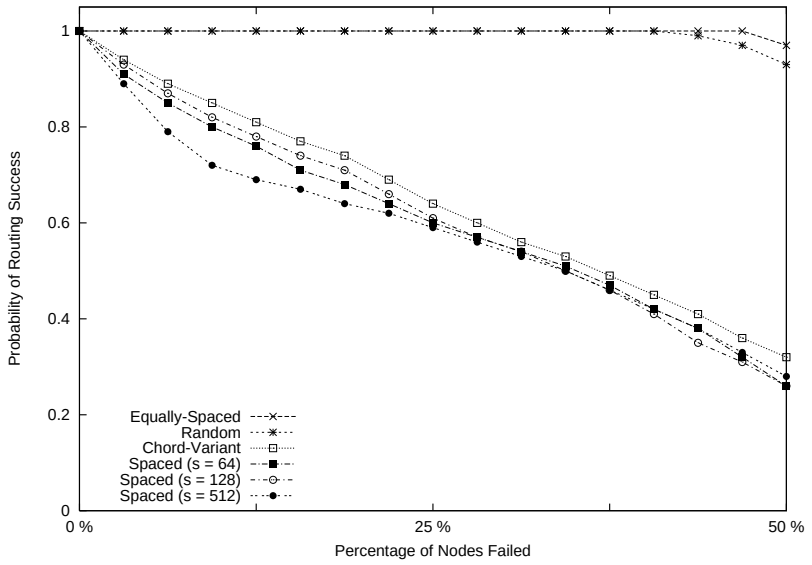


Figure 17: Probability of routing success in Chord with compromised runs ($N = 2^{20}, n = 1024, r = 4$).

replication degree (four replicas).

Furthermore, we investigated the possibility that an adversary may compromise a random subset of the nodes in the system. The results, shown in Figure 18, are astounding; when the adversary controls 25% of all nodes in the system, equally-spaced replication produces routes with no compromised nodes 98% of the time. Under the same level of attack, the Chord variant and spaced replication schemes successfully route only 25-60% of queries.

With only four equally-spaced replicas, we expect three disjoint routes. This implies that an adversary could prevent the success of a given query by compromising only three nodes. However, with far more nodes than that compromised in runs or randomly, almost all queries are resolved successfully. This result is extraordinary and confirms that route diversity does indeed improve routing robustness.

4.4 Summary and Discussion

To our knowledge, no significant work has been done to mitigate the effects of malicious routers in structured peer-to-peer systems using replica placement. We have

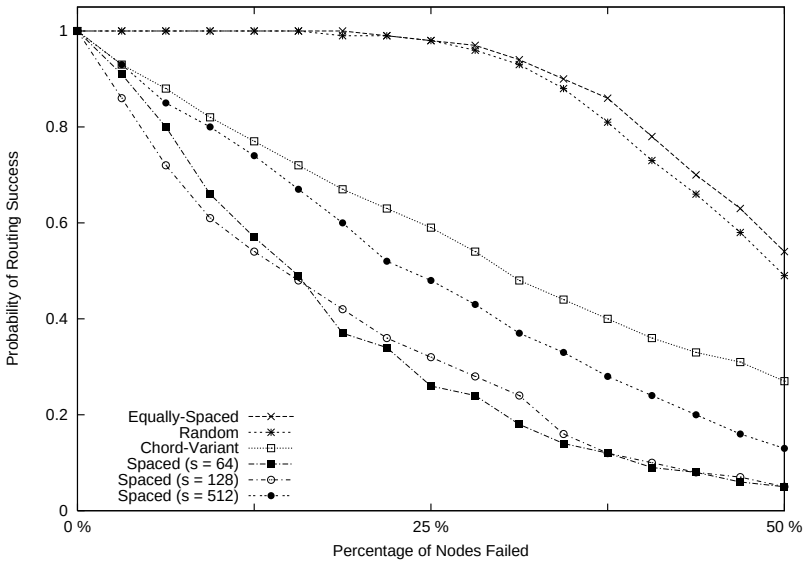


Figure 18: Probability of routing success in Chord with randomly compromised nodes ($N = 2^{20}, n = 1024, r = 4$).

shown that an equally-spaced replica placement scheme can improve routing robustness in Chord. To produce d disjoint routes per replica set using this scheme, each key must be replicated 2^{d-1} times. The solution requires minimal modification to the traditional Chord implementation and produces desirable results, even for sparsely populated rings. Our experiments have shown that equally-spaced replication can be used to successfully route 98% of queries in a ring that is 25% compromised.

A random replication scheme performs nearly as well as equally-spaced replication on average; however, the worst-case performance is significantly worse than equally-spaced placement. If an adversary is able to isolate them, these worst cases become an ideal target for attack because they are less fault tolerant. To the contrary, equally-spaced placement is consistently fault tolerant across all possible objects.

Equally-spaced replication is sufficient for improving routing robustness in Chord; however, we will show in Chapter 5 that there is a more general solution that can be applied to any distributed hash table that uses a tree-based routing scheme.

In this chapter, we have left some flexibility in the order and timing that replicas are queried. They may be queried in parallel, sequentially, or some combination of the

two. At first glance, the dispatch of all queries in parallel might seem optimal since the correct replica with the shortest route will be returned first. However, dispatching all of the lookups in parallel may put excessive load on the system thereby degrading overall performance. We consider this behavior briefly in Chapter 5. Furthermore, we will show in Chapter 6 that there can be economic benefits associated with querying replicas sequentially in order of decreasing cost.

CHAPTER V

REPLICA PLACEMENT IN TREE-BASED ROUTING DISTRIBUTED HASH TABLES

In this chapter, we expand the benefits of equally-spaced replica placement to a broader class of DHTs. We consider a class of DHTs that route messages using a scheme we call *tree-based routing*. We show that there exists a replica placement, which we call MaxDisjoint, that creates *disjoint routes* in DHTs of this type. We prescribe the number and placement of replicas necessary to produce d disjoint routes from any source node to the replica set. With this scheme, we are able to tolerate $d - 1$ malicious peers, whether they are acting to attack the storage and retrieval of data items or the routing infrastructure.

Using Pastry as an example, we evaluate MaxDisjoint through simulation and show that a DHT with typical configuration parameters can benefit from our replica placement. Our experiments show that with only eight replicas and a quarter of nodes compromised at random, a node can find a route, which consists of only uncompromised nodes, to a correct replica with greater than 97% probability. MaxDisjoint also tolerates runs of compromised nodes; with 16 replicas and 85% of the DHT compromised in a run, lookups can be resolved with greater than 96% probability. Furthermore, we use a technique we call *neighbor set routing* to increase route diversity and improve the probability of lookup success. For example, a lookup performed with neighbor set routing and MaxDisjoint placement can be resolved successfully with greater than 97% probability with 40% of nodes compromised at random. Finally, we demonstrate that the strategy used to query replicas can have a significant impact on performance and we propose a hybrid query strategy that can be used to

trade off response time and system load for the best performance.

5.1 Distributed Hash Tables with Tree-Based Routing

Distributed hash tables are often referenced by their geometry, i.e., ring (Chord [39]), torus (CAN [31]), tree (Plaxton [29]), or some hybrid (Pastry [32], Tapestry [43]). The geometry impacts neighbor and route selection, which can have an impact on flexibility, resilience, and proximity performance as studied in [17]. Although we use the term "tree-based routing", we are not referring to the geometry, but the routing algorithm. Tree-based routing algorithms have specific properties that we define herein.

5.1.1 Tree-Based Routing DHTs

Consider a distributed hash table (DHT) with an ordered id space I with size $N = |I|$ and a *branching factor* B such that $\log_B N$ is integral. The branching factor is used by each node to construct its routing table. The routing table of a node n has the following properties:

1. The node n partitions the entire id space into contiguous segments and selects one node from each id segment to include in its routing table. The partitioning is performed as follows:

   (a) The node n partitions the id space into B equal size contiguous parts.

   (b) Of the B id segments, n selects the id segment I' of which it is a member.

   (c) Steps 1a and 1b are repeated to re-partition I' until we have created parts of size one. The part that consists of the node n is discarded (there is no need for n to maintain a routing table entry to itself).

   (d) At the end of the partitioning process, n will have created $(B-1)\log_B N$ contiguous parts of the id space, $B-1$ each of sizes $\frac{N}{B}, \frac{N}{B^2}, \cdots, \frac{N}{B^{(\log_B N)-1}}$, and 1.

2. For each part P, n selects a node $p \in P$ that *covers* P and places it in its routing table. A node is said to cover a part P if its routing table contains $k > 1$ entries that cover the non-empty parts $P_1, P_2, \cdots, P_k$ and $P = P_1 \cup P_2 \cup \cdots \cup P_k$. By definition a node n is said to cover the part consisting of its id. This definition, combined with partitioning of P into non-empty parts, ensures that the recursive definition of coverage terminates.

The partitioning and routing table construction is shown graphically in Figure 19.

Note that tree-based routing DHT implementations differ in the manner in which Property 2 is satisfied. For example, in Chord [39], since routing is performed in the clockwise direction, the most counterclockwise node in each part must be selected because it is the only node that covers the entire id segment. In prefix-matching routing DHTs, all of the nodes within a part share a common prefix and cover the entire id segment; therefore, any node within the part may be chosen as a routing table entry. This explains the flexibility in choosing routing table entries in DHTs like Pastry [32] and Tapestry [43].

Routing is performed by forwarding the message destined for id the d to the entry that covers the id segment that contains d. We define a DHT constructed in this manner to be a *tree-based routing* DHT. If the paths from any source node to all possible destinations are aggregated, the resulting topology is a tree, which is how tree-based routing gets its name. Note that many popular DHT implementations [26, 29, 32, 39, 43] exhibit these properties and, therefore, employ a tree-based routing scheme. These DHTs have a number of useful properties, which we prove below.

First, tree-based routing DHTs are deterministic; that is, given a message destined for a node d, each node has one and only one routing table entry through which the message can be forwarded to d.

Lemma 2. *(Determinism) For the routing table of any node in a tree-based routing DHT, if the entries e_1 and e_2 cover id segments I_1 and I_2, respectively, then $I_1 \cap I_2 = \emptyset$.*

Proof. This follows naturally from the partitioning of the id space. □

Routes in tree-based routing DHTs are guaranteed to converge. This property holds when the DHT is *full*[1]; that is, every possible id is represented by a node.

Lemma 3. *(Routing Convergence) Consider the route in a full tree-based routing DHT from a source node s to a destination d represented as a series of nodes $n_1 = s, n_2, \cdots, n_{k-1}, n_k = d$ such that n_i is some entry e_j from the routing table of node n_{i-1} for $i > 1$. Suppose that $n_1, n_2, n_3, \cdots, n_k$ cover the id segments $I_1 = I, I_2, I_3, \cdots, I_k$[2]. Then, $I_k = \{d\} \subset I_{k-1} \subset I_{k-2} \subset \cdots \subset I_2 \subset I_1$.*

Proof. Consider the hop from n_j to n_{j+1}. Since the node n_j covers the id segment I_j, it must partition I_j into B equal sized id segments. Since n_{j+1} covers I_{j+1}, which is one of the parts of I_j, then $I_{j+1} \subset I_j$. Furthermore, since the DHT is full, the node n_{k-1} has a part that contains only the destination d. □

Furthermore, it is possible to bound the number of hops in every route; we state this formally below.

Lemma 4. *(Bounded Path Length) Any path in a full tree-based routing DHT has at most $\log_B N$ hops.*

Proof. Since the id segment covered by each hop must be a subset of the id segment covered by the previous hop, the minimum ratio between the size of id segments covered by consecutive hops is the branching factor B. Therefore, the longest path repeatedly divides N by B with each hop until it reaches the destination. This requires $\log_B N$ hops. □

[1] Some tree-based routing DHT implementations have to provide an additional mechanism to ensure routing convergence when the DHT is not full. Pastry, for example, maintains the neighborhood and leaf sets for this purpose.

[2] *Any node can cover the entire id space; therefore, we can state that n_1 covers I.*

5.1.2 Creating Disjoint Routes with Tree-Based Routing

Determinism and routing convergence provide a natural avenue for creating disjoint routes. Routing convergence guarantees that once a path enters a segment of the id space, it will never proceed to a node that is outside of that segment. If two paths can be created originating in different segments, then the paths are guaranteed to be disjoint. Furthermore, the determinism property ensures that any two routing table entries will route to different segments. Therefore, we can create disjoint routes simply by routing through different routing table entries. This is stated formally in the following lemma.

Lemma 5. *In a full tree-based routing DHT, routes originating at a common source node with different first hops are disjoint.*

Proof. Suppose two routes originating at a common source node n have first hops e_1 and e_2, such that e_1 and e_2 are different routing table entries of n. Using the determinism property, if e_1 and e_2 cover id segments I_1 and I_2, respectively, then $I_1 \cap I_2 = \emptyset$.

Consider any hops h_1 and h_2 in the routes beginning with first hops e_1 and e_2, respectively. Suppose hops h_1 and h_2 cover the id segments I_1' and I_2', respectively. Using the routing convergence property, $I_1' \subset I_1$ and $I_2' \subset I_2$. Since $I_1 \cap I_2 = \emptyset$, $I_1' \cap I_2' = \emptyset$ and, therefore, the routes are disjoint. □

Each routing table entry can be used as a first hop for every source node; therefore, we can state the number of disjoint routes that can be created from any source node by counting routing table entries.

Lemma 6. *In a full distributed hash table that employs tree-based routing, there are d disjoint routes from any source node, where $d = (B-1)\log_B N$.*

Proof. Employing Lemma 5, we can create a disjoint route for each of the d routing table entries by routing to a destination in the segment covered by each entry. We

cannot create $d + 1$ or more disjoint routes because two or more routes would share the first hop and overlap. □

The proof for Lemma 6 alludes to choosing multiple destinations to create disjoint routes. This lends itself naturally to a replica placement. In the following section, we propose a replica placement that creates disjoint routes, which we call MaxDisjoint.

5.2 The MaxDisjoint Replica Placement

Using Pastry as an example in the following sections, we will demonstrate how the properties of tree-based routing can be used to construct a replica placement that creates disjoint routes. We begin with an example placement to provide the reader with some intuition and then move toward a more formal definition. After defining the placement, which we call MaxDisjoint, we will evaluate the necessary replication degree to create a desired number of disjoint routes. Then, we will introduce the notion of a *run* and provide an expression for the maximum length run tolerable for a given replication degree. Next, we will discuss why MaxDisjoint is a more adaptive and flexibile solution than equally-spaced replication. Finally, we outline the basic elements of an implementation of the MaxDisjoint placement.

5.2.1 Intuition Behind MaxDisjoint Placement

To create route diversity in Pastry via replica placement, it is necessary to place replicas such that a given replica set will use a diverse set of routing table entries for every possible source node. We use an example to provide the necessary intuition. Consider a Pastry ring with id space of size 64 and prefix-matching in base-4 digits. We show the Pastry routing table structure graphically for a hypothetical node 121 in Figure 19.

Suppose we wish to replicate an object with id 101 in this Pastry ring. Node 121 routes to this object through the routing table entry marked “10x” in Figure 19.

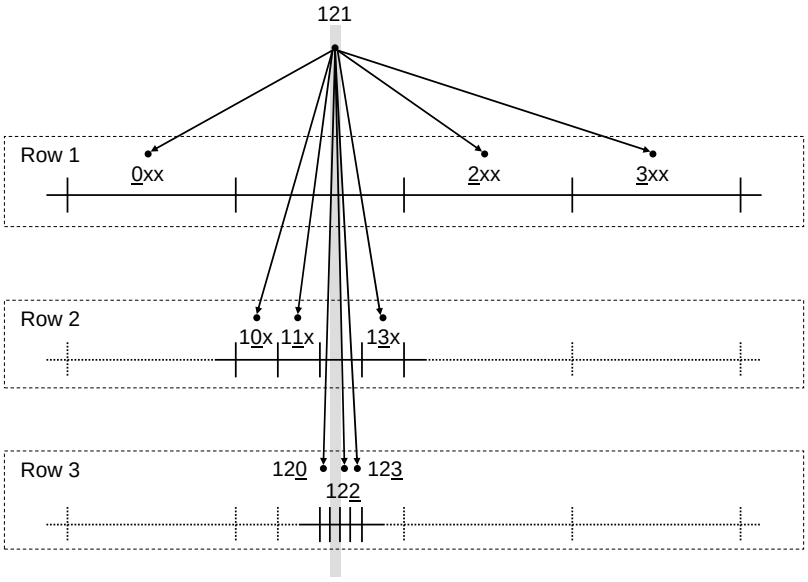


Figure 19: Pastry routing table structure for the hypothetical node 121 ($N = 64, b = 2$). The highlighted region marks the segment to which node 121 belongs.

Suppose we replicate the object with the id 111 to target the routing table entry "11x" in the example. This approach creates an additional disjoint route for any lookups for object 101 originating at node 121. One route is forwarded via the entry "10x" and the other is forwarded via "11x". However, consider another source node 221. This node routes to the object 101 and 111 through the same entry marked "1xx" and, therefore, does not gain an additional disjoint route.

To move toward a more effective approach, consider all the replicas of object 101 that would create an additional disjoint route for node 121. These are: 001, 111, 120, 122, 123, 131, 201, and 301. Note that there are total of nine possible disjoint routes[3] (including the route to the object 101), which is the number of routing table entries for node 121. Of these replicas, there are only three that can create an additional disjoint route for every possible source node: 001, 201, and 301. These replicas create disjoint routes by targeting entries in the first row of the routing table. Note that targeting an entry in the first row of the routing table requires a single replica whose

[3]There is actually a tenth "zero-hop" path that can be created by placing a replica at 121.

id differs from that of the master object in the first digit. To target entries deeper in the routing table, a larger number of replicas are required.

Suppose we wish to create five disjoint routes for all possible source nodes. Four routes can be created for every possible source node using the three replicas we have already discussed (001, 201, and 301) in addition to the object 101. To create the fifth route, we must target an entry deeper in the routing table. In the case of node 121, we may choose the replica 111. As alluded to before, this replica only creates a disjoint route for those source nodes whose ids start with the prefix "1" because these are the only nodes with an entry for "11x". Since there are four possible values for the prefix ($2^b = 4$), four replicas are required to target this routing table entry: 011, 111, 211, and 311. One of these four replicas will create an additional route for every possible source node depending on its prefix. The remaining three will be be routed through previously used routing table entries overlapping a previous route. This is shown graphically in Figure 20. Five disjoint routes are created for node 121, one each for the replicas R001 (or R011), R101, R111, R201 (or R211), and R301 (or R311). In a similar fashion, we can create a sixth disjoint route using the replicas 021, 121, 221, and 321; and a seventh using 031, 131, 231, and 331.

This pattern continues until the entire id space is exhausted. Note that in Pastry each node partitions the id space using prefixes and, therefore, we place replica by varying their prefixes. However, in general, we are simply spacing replicas such that replicas exist in different parts of each node's partitioning of the id space. In the next section, we provide an algorithm that generates these replica ids.

5.2.2 Definition of MaxDisjoint Placement

MaxDisjoint assigns each replica an identifier, which is used to determine its placement. The placement algorithm takes as input N, the identifier space size of the DHT; B, the branching factor; and d, the desired number of disjoint routes. We

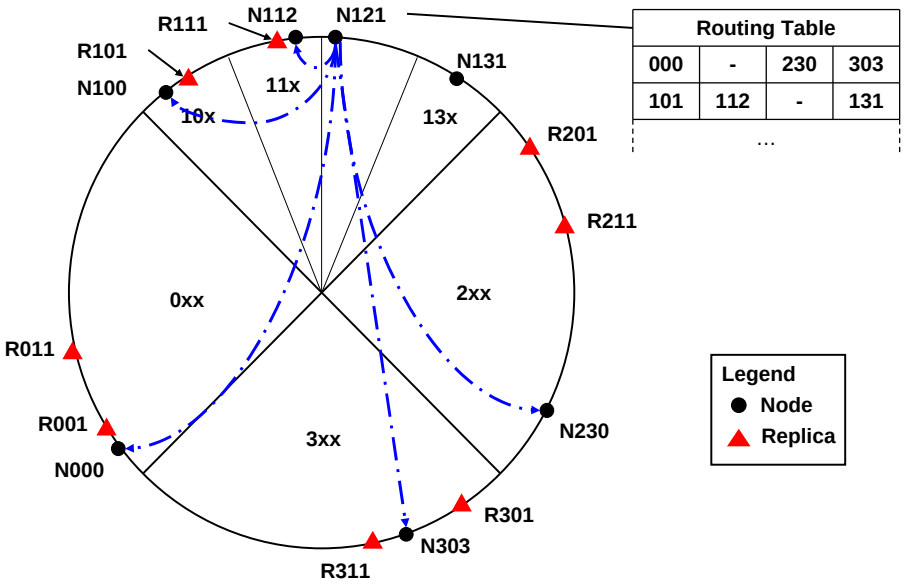


Figure 20: MaxDisjoint placement for object 101 to create five disjoint routes from query node 121. ($N = 64, b = 2$)

will prove that MaxDisjoint creates the desired number of disjoint routes in a later section.

Algorithm 1 (MaxDisjoint Replica Placement)**.** To create d disjoint routes, replicas are placed in $m+1$ rounds, where $m = \lfloor \frac{d-1}{B-1} \rfloor$. Each round consists of $B-1$ steps except for the final round, which consists of n steps, where $n = (d-1) \bmod (B-1)$. In the i-th round, B^{i-1} replicas are placed at equally spaced locations over the entire identifier space at each step. In step j of round i, the replica locations are given by:

$$R_{i,j} = \{k_{i,j},\ k_{i,j} + s_i,\ k_{i,j} + 2s_i, \ldots,\ k_{i,j} + (B^{i-1} - 1)s_i\} \pmod N,$$

where $k_{i,j} = k + j\frac{N}{B^i}$.

Looking at the example in Figure 20, consider the placement of an object 101 in a DHT with $B = 2^b = 4$ and $N = 64$. Suppose we want to create $d = 5$ disjoint routes. Then, we perform $m + 1 = 2$ rounds of the replica placement algorithm and $n = 1$ step in the final round. The replica locations and the corresponding rounds and steps of the algorithm are given below:

Round 1, Step 1:	201
Round 1, Step 2:	301
Round 1, Step 3:	001
Round 2, Step 1:	111, 211, 311, 011

As described in the replica placement algorithm, in each round replicas are placed starting at the master key and working in the direction of increasing identifiers. The algorithm is presented as such for its simplicity. However, the steps within each round can be performed in any order. Each step is functionally equivalent to the others in its round. Therefore, a real implementation may reorder the steps in each round to distribute the replicas more uniformly across the identifier space. This will help to provide load balance and tolerate runs of contiguous failed nodes. For instance, in the example above, to create two disjoint routes, we need only the master object (101) and one of the replicas created in round 1 (001, 201, or 301). Any of these replicas would create the second disjoint route, but choosing replica 301 creates a more uniformly distributed replica set.

5.2.3 Evaluation of Disjoint Routes

The desired number of disjoint routes d is one of the tunable inputs to the MaxDisjoint algorithm. Controlling the level of fault tolerance is an important design parameter and, therefore, d is a very useful input. In this section, we will formally prove that the algorithm indeed creates d disjoint routes in support of the intuition provided in earlier sections.

In our analysis, we assume that routing is performed in an identifier space of size N with branching factor B. All of our analytical results are proved within the context of a *full* DHT, but we will show, through experimentation, that these properties hold even in sparsely populated DHTs.

Our principle goal is to prove the following theorem:

Theorem 3. *The* MaxDisjoint *Algorithm produces* $d \leq (B-1)\log_B N$ *disjoint routes from any query node to a key* k *in a full tree-based routing DHT.*

Proof. Every set $R_{i,j}$ is a unique set of B^{i-1} replicas equally-spaced over the entire id space, which implies an inter-replica spacing of $\frac{N}{B^{i-1}}$. Consider a source node n and one of its routing table parts $P = \left[n, n + \frac{N}{B^{i-1}}\right)$. For each $R_{i,j}$, there exists one replica $r_k \in R_{i,j}$ such that $r_k \in P$. Furthermore, the replicas $\{r_k\}$ will be equally-spaced within P, separated by an inter-replica spacing of $\frac{N}{B^i}$. Regardless of how n chooses to partition P for its routing table, there exists a replica in each part of P. Therefore, there is a unique routing table entry for each replica r_k[4] and a disjoint route is created in every step of the algorithm. Using the definitions of m and n in the algorithm, it is easy to show that $d-1$ steps are performed. The $d-1$ disjoint routes created in the algorithm are combined with the route to k to create a total of d disjoint routes. □

As a corollary, we give the necessary replication degree to create d disjoint routes.

Corollary 1. *To produce* $d \leq (B-1)\log_B N$ *disjoint routes from any query node to a key* k *in a full tree-based routing DHT with branching factor* $B > 1$*, the key* k *must be replicated at* $(n+1)B^m$ *locations determined by the* MaxDisjoint *Algorithm, where* $m = \lfloor \frac{d-1}{B-1} \rfloor$ *and* $n = (d-1) \bmod (B-1)$.

Proof. Since we have proved that d disjoint routes are indeed created by the algorithm, we need to show that performing the algorithm with input d places a copy of k at $(n+1)B^m$ locations in the id space. The key k accounts for one location and the remaining locations are determined by the replica placement. Since, B^{i-1} replicas are

[4]It is possible that one of the replicas is placed at n. Even though no routing is performed to fetch this replica, we count it as a route of zero hops.

placed at each step in round i, the total number of replicas is given by:

$$\begin{array}{rcccccc} & \text{(key)} & + & \left(\begin{array}{c}\text{replicas placed in}\\ \text{first } m \text{ rounds}\end{array}\right) & + & \left(\begin{array}{c}\text{replicas placed in}\\ \text{final } n \text{ steps}\end{array}\right) \\ = & 1 & + & \sum_{i=1}^{m}(B-1)B^{i-1} & + & nB^m \\ = & (n+1)B^m & & & & \end{array}$$

□

We give our replica placement the name MaxDisjoint because Corollary 1 prescribes the minimum replication degree to create the desired number of disjoint routes using this placement. We state this formally in the following theorem.

Theorem 4. *To produce d disjoint routes from any query node to a key k in a tree-based routing DHT with $B > 1$, the key k must be replicated at no fewer than $(n+1)B^m$ locations determined by the* MaxDisjoint *Algorithm, where $m = \lfloor \frac{d-1}{B-1} \rfloor$ and $n = (d-1) \bmod (B-1)$.*

Proof. Assume d disjoint routes can be created with $(n+1)B^m - 1$ locations determined by the MaxDisjoint Algorithm, where $m = \lfloor \frac{d-1}{B-1} \rfloor$ and $n = (d-1) \bmod (B-1)$. In other words, d disjoint routes are created with one fewer replica than prescribed by Theorem 3. Let r be the missing replica. We will show that there exists a query node q for which d disjoint routes are not created. Suppose $r \in R_{i,j}$, then let q be a node in $\left[r, r + j\frac{N}{B^i}\right)$. The replica r is the only replica in $R_{i,j}$ that can create a disjoint route for the query node q. Therefore, step j in round i does not create a disjoint route and we cannot form d disjoint routes with one fewer replica. □

It is worth noting that MaxDisjoint provides these properties without modifying the underlying routing mechanism. MaxDisjoint naturally creates disjoint routes using the properties of the tree-based routing scheme.

5.2.4 Chord as a Tree-Based Routing DHT

To provide consistency with Chapter 4, we reconsider Chord as a tree-based routing DHT. It is straightforward to show that Chord finger tables are constructed like tree-based routing tables with $B = 2$. Therefore, we can apply Theorem 3 to give the following corollary.

Corollary 2. *To produce $d \leq \log_B N$ disjoint routes from any query node to a key k in a full tree-based routing DHT with branching factor $B = 2$, the key k must be replicated at 2^{d-1} equally-spaced locations in the ring.*

Proof. When $B = 2$, $m = d - 1$ and $n = 0$. Therefore, d disjoint routes are created by replicating k at 2^{d-1} locations in the ring. Furthermore, round i will place 2^{i-1} replicas equally-spaced over the id space. Aggregating the replicas placed in each round will result in 2^{d-1} replicas equally-spaced over the id space with inter-replica spacing $\frac{N}{2^{d-1}}$. □

Note that the claim in Corollary 2 is consistent with the findings in Chapter 4.

5.2.5 Toleration of Runs

We define a run of length l starting at peer m to be the contiguous set of peers with identifiers in the interval $[m, m + l)$. As indicated in [39], an adversary can create imbalance in the distribution of peers in the DHT by appropriately selecting identifiers. In the worst case, an adversary can take control of a contiguous sequence of identifiers or, using our terminology, a run of peers.

We claim that MaxDisjoint replication can tolerate adversarial runs of bounded length in tree-based routing DHTs. Before proving the tolerable length of a run, we provide some intuition of how a run may be used to disrupt routing in the DHT. Consider a query node q. An adversary can reduce the number of replicas reachable from q by $\frac{1}{B}$ by controlling the peer with identifier $q + \frac{N}{B}$, where N is the identifier

space size and prefix-matching is performed in base B. This is because all of the replicas in the interval $[q + \frac{N}{B}, q + 2\frac{N}{B})$ are routed through the node $q + \frac{n}{B}$ ($\frac{1}{B}$ of all replicas are in this interval). If the adversary controls a run of nodes ending at $q + \frac{B-1}{B}\frac{N}{B}$, he can control a larger number of replicas.

Theorem 5. *A full tree-based routing DHT with an identifier space of size N, and $r \geq B$ replicas placed using* MaxDisjoint *placement can tolerate any adversarial run of length $1 + N(\frac{B-1}{B} - \frac{1}{B^m})$, where $m = \lfloor \log_B r \rfloor$.*

Proof. (by induction) Since we are using MaxDisjoint placement, it is straightforward to show that $m = \lfloor \log_B r \rfloor = \lfloor \frac{d-1}{B-1} \rfloor$. This implies that the maximum length run tolerable by the DHT changes with each round of the MaxDisjoint algorithm.

Consider a peer q. For $B \leq r < B^2$, $m = 1$ and there exists a replica k in the interval $[q, q+\frac{N}{B})$. If we assume that the adversary has control of the peer $q+(B-1)\frac{N}{B}$ (which is the worst case), then we must ensure that k is not in the run. Thus, the maximum length run is $[q + \frac{N}{B}, q + (B-1)\frac{N}{B})$, which has length $(B-1)\frac{N}{B} - \frac{N}{B} + 1$ or $1 + N(\frac{B-1}{B} - \frac{1}{B})$.

Assume that the maximum length run tolerable with r replicas is $1+N(\frac{B-1}{B}-\frac{1}{B^m})$, where $m = \lfloor \log_B r \rfloor$. Consider a query node q. If we assume the adversary takes control of the peer $q + (B-1)\frac{N}{B}$, then he does not control any peers in the interval $[q, q+\frac{N}{B^m})$. Furthermore, there must be at least one replica in this interval. If another round of the MaxDisjoint algorithm is performed, then there will be B replicas in this interval separated by a spacing of $\frac{N}{B^{m+1}}$. Thus, the length of the tolerable run increases by $\frac{N}{B^{m+1}}$ to $1 + N(\frac{B-1}{B} - \frac{1}{B^{m+1}})$. □

In a Pastry DHT with a typical value of $B = 16$ and 16 replicas, an adversary may control a run of more than 85% of the identifier space and there will be a route to at least one replica for every possible query. As the replication degree approaches the identifier space size, the maximum length run tolerable by the DHT approaches

$\frac{B-1}{B}N$.

5.2.6 Adaptivity and Flexibility of MaxDisjoint

There is a noted similarity between the MaxDisjoint and equally-spaced placements. In fact, when $n = 0$, MaxDisjoint produces equally spaced replica locations. One may argue that equally-spaced replica placement is as effective as MaxDisjoint in creating disjoint routes. However, replica placements have other desirable properties other than their ability to create route diversity; equally-spaced placement fails to deliver some of these benefits when $B > 2$.

Two properties in particular are adaptivity and flexibility. Adaptivity is the ability to easily change the replication degree of an object without incurring a large overhead. If the replication degree of an object is changed, we would like to minimize the number of messages exchanged and objects shifted. Flexibility is the ability to easily vary the replication degree of different objects without incurring a large overhead. Certainly, some objects may be more popular or critical than others and require a higher degree of replication. The placement should replicate objects to varying degrees without using excessive time or state at the time of insertion and lookup.

MaxDisjoint provides adaptivity more effectively than an equally spaced placement. A change in the replication degree must be handled carefully to maintain equal spacing. Consider an increase in the replication degree to add an additional disjoint route. With MaxDisjoint, the additional replicas are placed at the locations determined by performing another step in the placement algorithm leaving the existing replicas in their current locations.

With equally spaced replicas, additional replicas can be introduced in two different ways. The first option is to compute the equally-spaced replica locations for the new replication degree and shift existing replicas, if necessary. In some cases, the existing replica locations will not be a subset of the new replica locations. This implies that

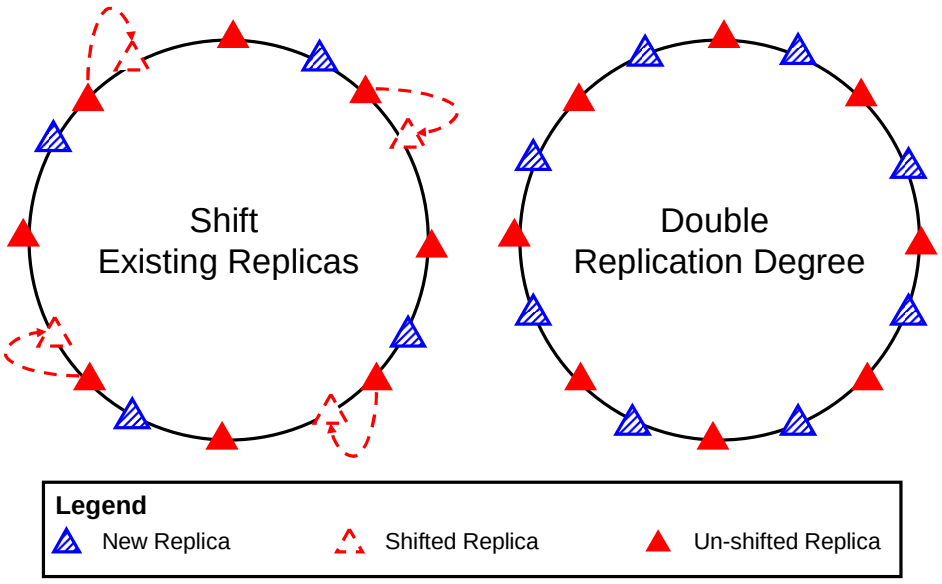


Figure 21: Two methods for increasing the replication degree when using equally spaced replica placement.

existing replicas will have to be shifted, which has a non-negligible cost. The second option is to double the current replication degree such that no existing replicas must be shifted. These two options are depicted graphically in Figure 21.

Doubling the replication degree avoids the cost of shifting existing replicas, but may come with the added burden of storing an excessive number of replicas. The number of replicas prescribed by Theorem 3 is sufficient. For example, in Figure 21, only four additional replicas are needed to create an additional disjoint route; however, doubling the replication degree introduces eight new replicas. The excess storage burden created when doubling the replication degree is shown quantitatively for $B = 16$ in Figure 22. MaxDisjoint is able to create a desired number of disjoint routes more effectively than equally-spaced placement when there is a change in the replication degree.

A sound replica placement also provides flexibility; that is, the replication degree of one data item is not dependent on the replication degree of any other data item. Fortunately, flexibility can be provided easily by MaxDisjoint placement.

The problem of flexibility arises at the time of lookup. Without knowledge of

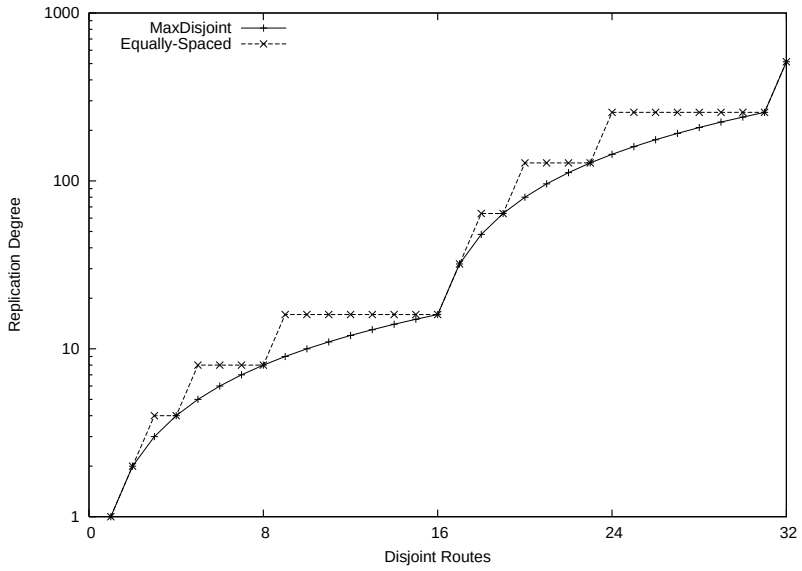


Figure 22: Replication degree for increasing numbers of disjoint routes.($B = 16$)

the replication degree of the target, the replica locations cannot be determined. As a solution, rather than fixing a system-wide replication degree for all data items or storing the replication degrees for all objects, we fix a maximum replication degree r_{max} for all objects. For a data item with replication degree $r \leq r_{max}$, the r replicas will be located at a subset of the r_{max} replica locations. As a result, for some lookups, we may query a replica that does not exist, but we trade off these extra messages for the reduction in node state.

5.2.7 Implementation

To uniquely identify each replica, we use a *key identifier pair* (k, v), where k is the *key identifier* and v is *virtual key identifier*. For each replica, v gives the location of the master key. By definition, the master key k is denoted by the pair (k, k). We require an ordered pair because two data items may have replicas that reside on the same peer.

When a key k is inserted into the DHT with replication degree d, we first compute the key identifier pairs for each replica: $(k, k), (k_1, k), (k_2, k), \ldots (k_{d-1}, k)$. Once the

key identifier pair for each replica is computed, we use the DHT key insertion mechanism to insert the replicas. That is, we perform a lookup for each key identifier and store the replica in their respective locations.

Next, the DHT lookup primitive must be modified to accommodate the new replication scheme. When a peer is queried for a key, the query node must compute the locations of all replicas. The key identifier is used to route to the replicas. Once a replica's home node is found, the key identifier pairs are compared to return the appropriate replica.

It is worth noting that no additional routing table entries are required to route to the replicas. In addition, if the query node dispatches the lookups for the entire replica set simultaneously, there may be an improvement in performance because the query node can return the first correct response received (which may have returned along a route shorter than the route to the master key). However, if the added load of the extra lookups puts strain on the system, the performance may improve only slightly or even degrade. We evaluate this hypothesis through experimentation.

Finally, when peers join or leave the DHT, the DHT join and leave mechanisms can be used by simply ignoring the virtual peer identifiers in each key identifier pair. Note that DHT replica placement schemes that place replicas in the neighborhood of the master key require modification to the node join and leave mechanisms. To maintain the replication degree, replicas will need to be shifted for every join or leave. Ghodsi et al. [16] discuss the effect of churn on symmetric (equally-spaced, MaxDisjoint) replication and show that only $O(1)$ messages are needed to maintain the replication degree for every join or leave compared to $\Omega(r)$ messages for a successor-list (neighbor set) placement, where r is the replication degree.

5.3 Experiments

To confirm that our analytical results hold for sparsely populated DHTs or DHTs with clustered id spaces, we conducted a series of experiments through simulation. First, to measure the impact of replica placement on routing robustness, we consider the number of disjoint routes created for several replica placements. Furthermore, we find the probability of lookup success when nodes are compromised at random or in a run of several nodes.

Second, we consider a heuristic used for creating route diversity in [7] that we call *neighbor set routing*. We measure its ability to create route diversity and the impact on the probability of lookup success.

Finally, having shown that replica placement can improve on routing robustness, we consider the impact of parallel queries on response time.

Simulation Environment All experiments were performed using a Java-based simulator we have developed. We are able to model Chord and Pastry routing, uniform and clustered node distributions, and two adversarial models. Nodes may be compromised at random with some failure probability or in a run of several nodes. The simulator is extensible to model other DHT implementations, node distributions, and adversarial models.

Each data point in our results is representative of over 100,000 lookups performed in 10 different random node distributions. We simulate a lookup by randomly selecting an uncompromised query node and a target key. In reality, if the query node is compromised, it can affect the outcome of the lookup. However, if we assume that data items are self-verifying, a compromised query node can only cause the client to timeout and select another query node. We deem a lookup successful if there exists a route consisting of only uncompromised nodes from the query node to any replica of the target key. If all routes from the query node to the replica set contain

Table 2: Pastry simulation parameters.

Symbol	Parameter
N	Identifier Space Size
n	Number of Nodes
B	Branching Factor (2^b for Pastry)
r	Replication Degree
f	Fraction of Compromised Nodes
C	Number of Clusters
σ	Cluster Width (standard deviation)

a compromised node, then the lookup is deemed to fail.

For most experiments it is sufficient to compute routes in the network using the appropriate routing protocol. However, to measure response time, it was necessary to modify our simulator to be event-driven. The remaining simulation parameters are summarized in Table 2.

Replica Placements Considered In our experiments, we consider four replica placements: MaxDisjoint; *neighbor set*, where replicas are placed at distinct nodes in the neighborhood of the root (e.g., Chord successor list, Pastry leaf set); *random*, where replica locations are uniformly distributed; and *spaced*, where replica identifiers are separated by a uniform spacing s. It is worth noting that two replicas may be placed at the same node with spaced replication.

In the case of neighbor set placement, some implementations may attempt to reduce load by querying the entire replica set with a single lookup message. This naturally creates route overlap; for a fair assessment, we dispatch a separate lookup for each replica in the replica set.

5.3.1 Measurement of Disjoint Routes

First, to verify the correctness of our analysis, we measure the average number of disjoint routes created using the considered replica placements. These results are

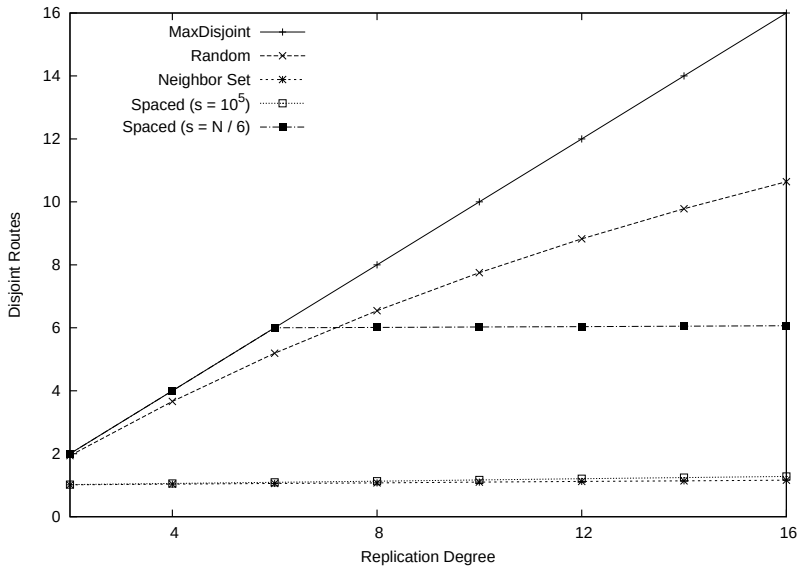


Figure 23: Number of disjoint routes with increasing replication degree in Pastry. ($N = 2^{28}, n = 8192, B = 16$)

depicted in Figure 23. For the parameters tested, MaxDisjoint placement outperforms the other placements in creating disjoint routes.

The neighbor set placement does not create a significant number of disjoint routes as expected. Routes toward keys that are close to each other in the identifier space are likely to converge. Since the neighbor set placement clusters replicas, an adversary can eliminate the entire replica set if he can compromise a node common to all routes destined for that neighborhood. By increasing the route diversity, we eliminate these single points of vulnerability.

The performance of the spaced placement scheme is dependent on the spacing chosen. If the spacing is small, then the spaced placement is very similar to the neighbor set placement. In the extreme case, if the replica spacing is much less than the average inter-node spacing, two or more replicas may be placed at the same node. We observe this phenomenon for the spaced placement with $s = 10^5$ in Figure 23. As we increase the spacing, there is a tendency to increase the number of disjoint routes. However, as we increase the replication degree, the replica locations will wrap around the identifier space and no additional disjoint routes will be created. We observe

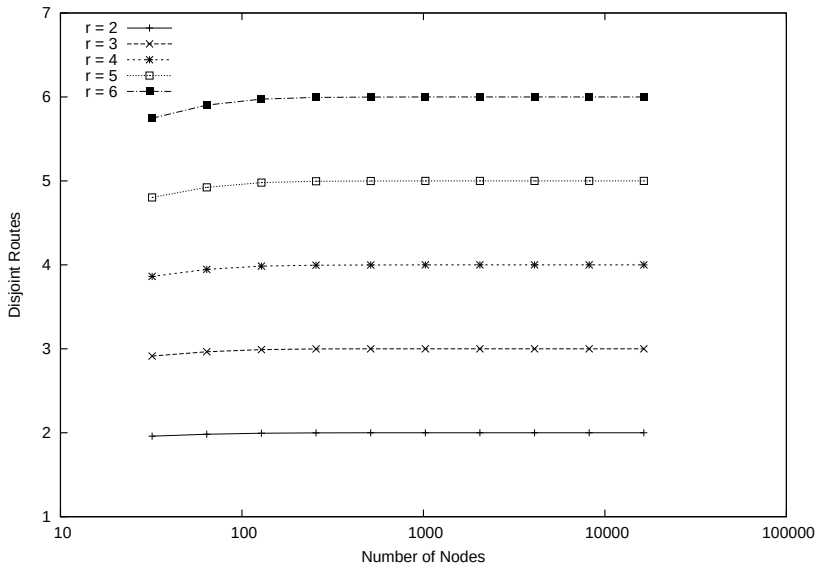


Figure 24: Number of disjoint routes with increasing number of nodes in Pastry. ($N = 2^{28}, B = 16, d \in \{2, 3, 4, 5, 6\}$)

this phenomenon when the spacing $s = N/6$. This implies that the spacing should be a function of the replication degree, which is fundamental to how MaxDisjoint creates disjoint routes.

The random placement creates nearly as many disjoint routes on average because replicas are uniformly distributed. However, there is significant difference between MaxDisjoint and random placement in the worst case. We present an argument in support of this claim at the end of this section.

To ensure that this number of disjoint routes could be created in more sparsely populated id spaces, we varied the number of nodes in the DHT starting from 32 and measured the number of disjoint routes for various replication degrees. This data is depicted in Figure 24. The actual number of disjoint routes converges quickly to the theoretical value, which implies that MaxDisjoint placement can be used effectively in very sparsely populated networks. In these results, the theoretical number of disjoint routes is achieved for loads greater than 1000 nodes, which is less than a thousandth of a percent of the identifier space.

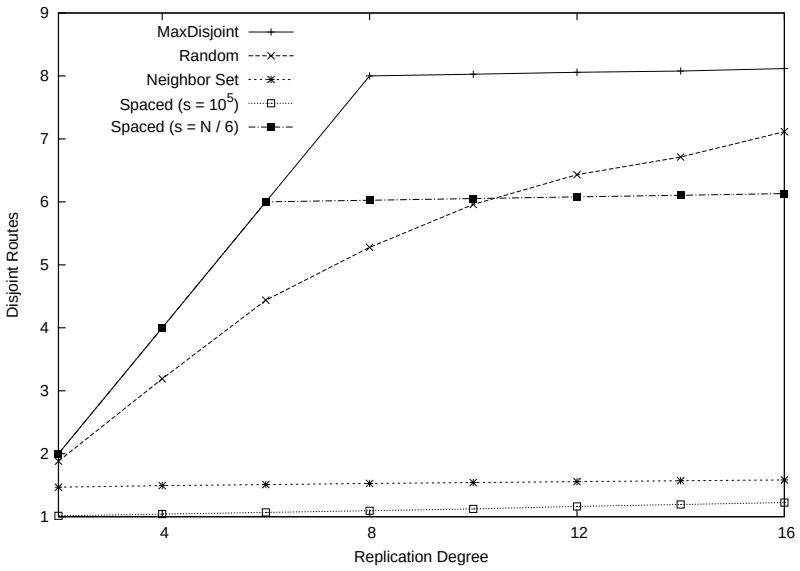


Figure 25: Disjoint routes for tightly clustered nodes in Pastry ($N = 2^{28}, n = 8192, \sigma = 2^{15}$).

5.3.2 Resilience to Node Clustering

To model the population distribution that may result from the collusion of several malicious nodes, a series of experiments were run on clustered DHTs. To model a clustered distribution, four node ids were randomly selected as cluster means such that the clusters are non-overlapping and unpopulated gaps exist in the id space. The cluster density was tuned using the standard deviation of the Gaussian distribution centered around each cluster mean. The number of disjoint routes created for $\sigma = 2^{15}$ and $\sigma = 2^{16}$ are shown in Figures 25 and 26, respectively.

Clustering can marginally reduce the number of disjoint routes created for small replication degrees, but the impact is more dramatic when creating a larger number of disjoint rotues. The impact of clustering is intensified with tighter clusters (i.e., decreasing σ) because replicas are not located at the positions that MaxDisjoint prescribes. Although the replica ids are properly assigned, the replicas themselves are confined to the clusters. One or more of the replicas may lay in the unpopulated gaps between clusters. These replicas get "pushed" to next cluster in the id space,

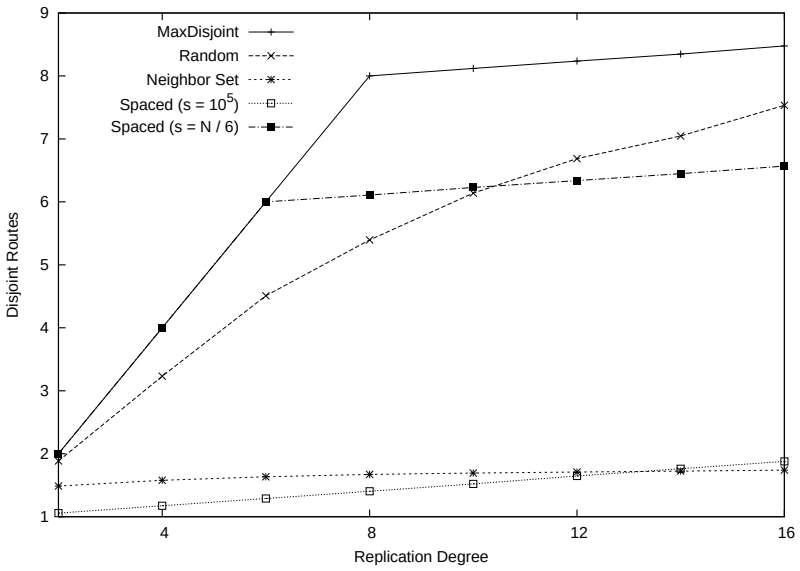


Figure 26: Disjoint routes for loosely clustered nodes in Pastry ($N = 2^{28}, n = 8192, \sigma = 2^{16}$).

possibly eliminating a disjoint route. This is more likely to happen when creating a large number of disjoint routes (which requires more replicas).

We computed the percentage decrease in disjoint routes that results from clustering. This analysis produced some very interesting results that are depicted in Figure 27. One interesting conclusion is that not all replica placements are negatively affected by node clustering. In particular, when replicas are placed close together in the id space, e.g., neighbor set or spaced (for small spacings) placement, there can actually be an increase in the number of disjoint routes created. This is because a long string of closely packed replicas may span a gap between clusters, which pushes some of the replicas to another cluster. The clustering actually helps distribute the replicas better resulting in more disjoint routes. Nevertheless, the 4-6% increase in the number of disjoint routes is insubstantial because these placements fail to create a sufficient number of disjoint routes with uniformly distributed nodes.

To the contrary, the MaxDisjoint and random placements are affected negatively by clustering. Placements that distribute replicas in the id space may place a replica in the unpopulated gaps between clusters. These replicas get pushed to a cluster

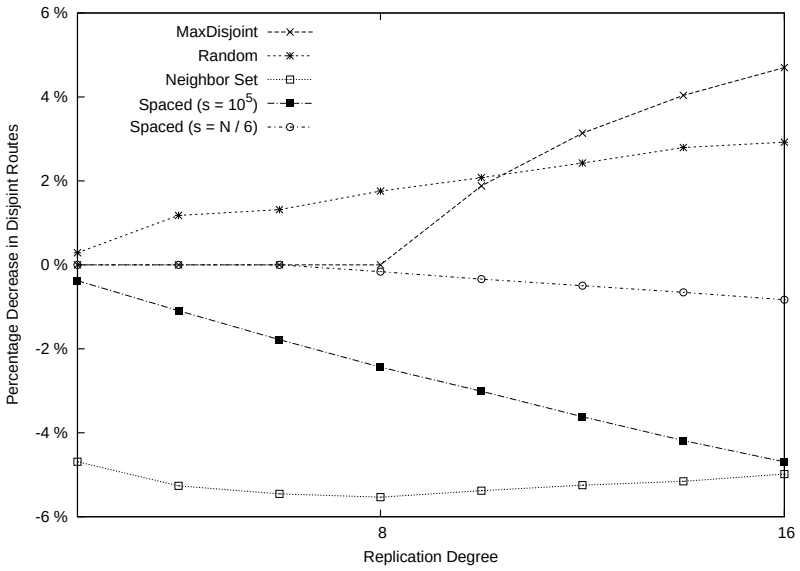


Figure 27: Percent decrease in disjoint routes from uniformly distributed nodes to a clustered distribution in Pastry ($N = 2^{28}, n = 8192, C = 4, \sigma = 10^{16}$).

and a disjoint route may be lost. This phenomenon takes effect for MaxDisjoint when $r = 8$. At this point, the inter-replica spacing is 2^{25}. If we use two standard deviations to capture 95% of each cluster, we can estimate the inter-cluster gaps to about 2^{26}, which is twice the inter-replica spacing. That implies that about half of all replicas are located in the gaps between clusters. Note that the random placement suffers from this problem over the entire range of replication degrees because it is not guaranteed to distribute replicas across the entire id space. Furthermore, as we increase the replication degree, more replicas will be located in the gaps and we lose the benefit of increased replication degree. Nevertheless, the 3-5% decrease in the number of disjoint is relatively insignificant because MaxDisjoint creates so many more disjoint routes than the other placements in a uniformly populated id space.

5.3.3 Impact of Replica Placement on Routing Robustness

To demonstrate that the number of disjoint routes has a significant impact on the robustness of the DHT, we measure the probability of lookup success with a random fraction of faulty nodes. A faulty node may be a failed or compromised node. The

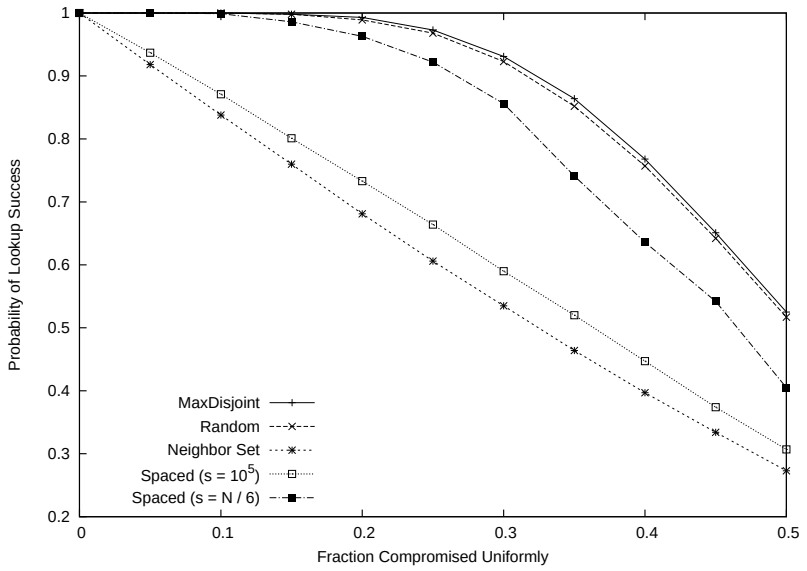


Figure 28: Probability of routing success with uniformly compromised nodes in Pastry. ($N = 2^{28}, n = 8192, B = 16, r = 8$)

probability of routing success is shown in Figure 28.

These results indicate a correlation between the number of disjoint routes and the probability of lookup success. The MaxDisjoint and random placements most effectively create disjoint routes and, thus, have the most positive impact on the probability of routing success. Furthermore, we can conclude that using MaxDisjoint placement instead of neighbor set placement can dramatically improve the probability of routing success. With a quarter of nodes compromised at random, greater than 97% of all lookups can be resolved successfully with MaxDisjoint placement compared to only 60% with neighbor set placement.

With the network configuration in Figure 28, the MaxDisjoint placement creates eight disjoint routes. Therefore, an adversary could prevent the correct resolution of a given query by compromising only eight nodes. However, with far more nodes than that compromised at random, nearly all queries are resolved successfully. This is a strong indication that replica placement plays a critical role in providing robustness in distributed hash tables.

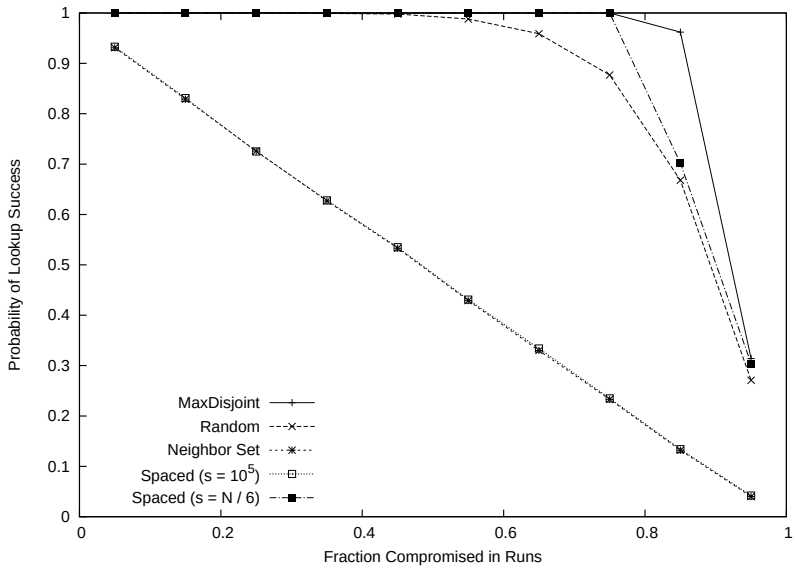


Figure 29: Probability of routing success with runs of compromised nodes in Pastry. ($N = 2^{28}, n = 8192, B = 16, r = 16$)

Our analysis indicates that MaxDisjoint should be able to tolerate runs of compromised nodes better than placements that cluster replicas closely together. Experimental results that confirm this hypothesis are depicted in Figure 29. With 16 replicas and 85% of the identifier spaced compromised in a run, greater than 96% of lookups are resolved successfully with MaxDisjoint placement compared to only 13% with neighbor set placement. Furthermore, these results demonstrate an exploit of random placement. With moderately long runs, MaxDisjoint is able to successfully resolve a higher fraction of queries than the random placement. For example, with 85% of the identifier spaced compromised in a run, only 66% of lookups are resolved successfully with a random placement. This is because the random placement may cluster replicas for a significant fraction of keys. We investigate this further in the following section.

MaxDisjoint versus Random Placement In the experimental data presented thus far, random placement seems to have performed on par with MaxDisjoint on average. We argue, however, that a truly random placement is expensive to implement

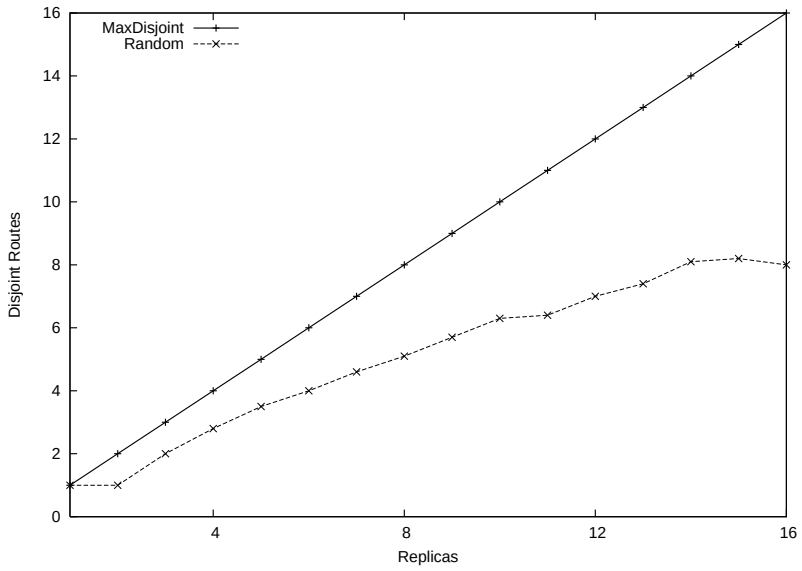


Figure 30: Worst-case performance of MaxDisjoint and random replica placement in Pastry. The minimum number of disjoint routes created over all lookups is shown. ($N = 2^{20}, n = 8192$)

in practice and the average performance of a random placement does not trickle down to the worst case.

The worst case attack against a replica placement is to target the replica locations themselves. One may argue that a placement with random locations would make it more difficult for an adversary to determine and target the replica set than a deterministic approach like MaxDisjoint. However, a truly random placement also complicates matters for the query node. The query node must be able to compute the replica locations for any object at the time of lookup. To avoid keeping a considerable amount of state at each node, we may use multiple hash functions to generate random replica locations. However, this implementation is not truly random and, with knowledge of the hash functions, it is as vulnerable as MaxDisjoint.

As far as performance is concerned, the results we have shown depict the average number of disjoint routes created. To consider the worst case performance, we measured the minimum number of disjoint routes created for MaxDisjoint and random placement. These results are depicted in Figure 30.

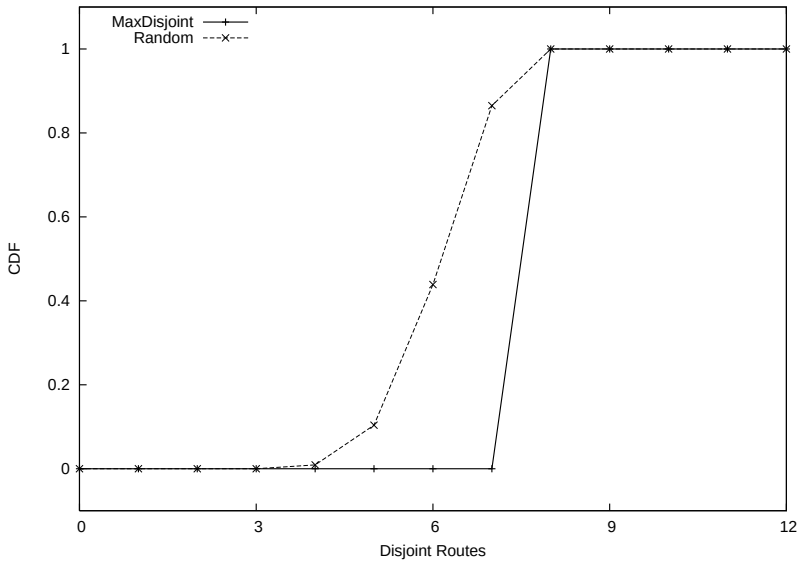


Figure 31: Cumulative distribution of disjoint routes created for MaxDisjoint and random placement in Pastry. ($N = 2^{20}, n = 8192, r = 8$)

The results in Figure 30 confirm our hypothesis of the worst case performance of a random placement. For a few objects, the placement may create as few as half of the desired number of disjoint routes. To establish that the worst case was not an isolated case, we measured the number of disjoint routes created over all lookups. The cumulative distribution of the number of disjoint routes with eight replicas is shown in Figure 31.

For the case depicted in Figure 31, MaxDisjoint created the expected eight disjoint routes for every single lookup. To the contrary, the random placement creates six or fewer disjoint routes for 45% of lookups. We observed the same behavior in other cases as well. Therefore, if delivering a consistent level of fault tolerance across all lookups is a design constraint, random placement is not a reasonable solution.

5.3.4 Replica Placement and Neighbor Set Routing

We believe replica placement is an efficient way of creating disjoint routes because it does not require significant modification to the underlying DHT routing scheme. Other works have considered reusing the existing routing information to create route

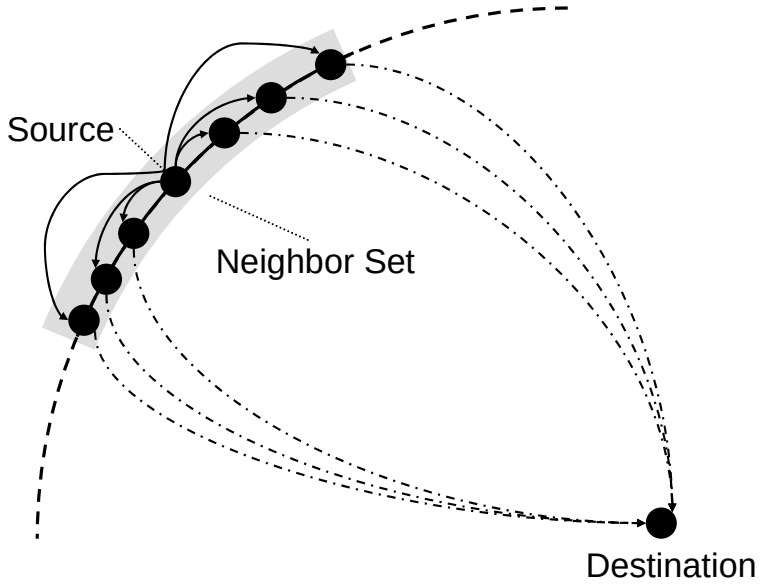


Figure 32: Graphical depiction of neighbor set routing.

diversity [7, 30]. Although these approaches do not create provably disjoint routes, we believe there is value in introducing some additional form of route diversity. Furthermore, we believe these techniques could be combined with our replica placement to provide additional benefit. To evaluate this claim, we consider the route diversity technique introduced by Castro et al. [7], which we call *neighbor set routing*.

Castro et al. use neighbor set routing is used to find diverse routes toward the neighborhood of a key. To create diverse routes, messages are routed via the neighbors of the source node. This is depicted graphical in Figure 32. Castro et al. claim this technique is sufficient in the case when replicas are distributed uniformly over the identifier space, as in CAN and Tapestry. We consider the ability of neighbor set routing to create diverse routes to a replica to enhance the routing robustness of MaxDisjoint.

To evaluate the relative impact of replica placement and neighbor set routing, we measured the probability of routing success for four scenarios:

1. Neither replication nor neighbor set routing is used,

2. Only neighbor set routing is used by routing through eight neighbors,

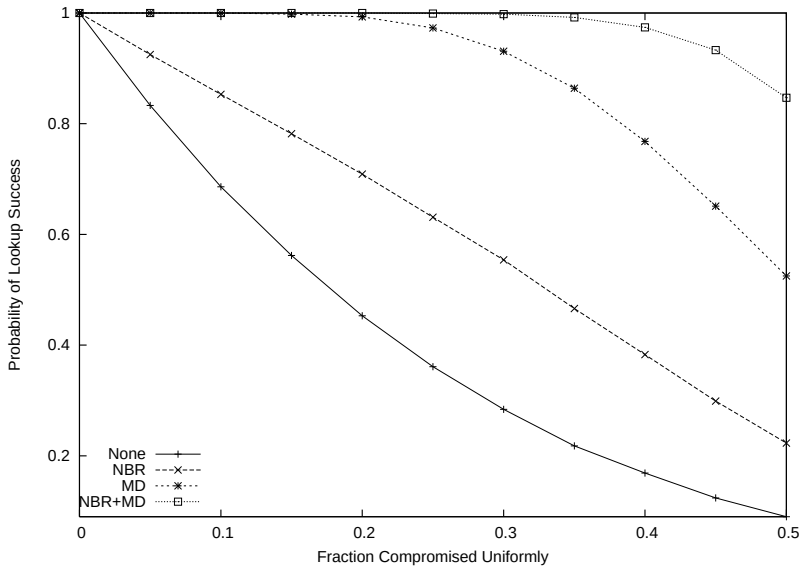


Figure 33: Probability of routing success with neither replication nor neighbor set routing (None), neighbor set routing only (NBR), MaxDisjoint placement only (MD), and both neighbor set routing and MaxDisjoint placement together (NBR+MD). ($N = 2^{28}, n = 8192, B = 16, r = 8$)

3. Only MaxDisjoint placement is used with a replication degree of eight, and

4. Both neighbor set routing and MaxDisjoint placement are used by routing messages through eight neighbors to eight replicas.

These results are depicted in Figure 33.

Both MaxDisjoint placement and neighbor set routing can have a positive impact on the probability of lookup success. However, the trend is stronger with replica placement. With a quarter of nodes compromised at random, over 97% of lookups are resolved successfully with MaxDisjoint placement compared to 63% with neighbor set routing alone. At best, neighbor set routing can create independent routes, since all paths will converge at the destination. If the destination node is compromised, no amount of route diversity can increase the probability of lookup success.

Nonetheless, the added route diversity that neighbor set routing provides can benefit MaxDisjoint placement, especially with a large fraction of compromised nodes. With 50% of nodes compromised, the probability of lookup success of using

MaxDisjoint placement improves from 52% to 84% when combined with neighbor set routing.

5.3.5 Parallel Queries and Response Time

Finally, since replica placement seems to be a reasonable method for improving routing robustness, it is natural to consider some practical concerns of using replication. When querying a replica set, response time may be reduced by querying the entire replica set in parallel. Alternatively, this could have a significant impact on the system load, which may result in congestion and increased response time. Therefore, we consider three replica query strategies: parallel, sequential and hybrid.

Unlike the parallel strategy, the sequential strategy queries replicas one at a time waiting for a response before querying the next replica. This strategy controls system load at the expense of response time. With very few failures, the sequential strategy should result in reasonable response times without inducing a significant load on the network. However, the response time may not be as resilient to an increase in failure rate as the parallel strategy.

We also consider a hybrid approach in which replicas are queried in sets of two or more replicas. Sets are queried one at a time waiting for a response before querying the next set. With this strategy, the trade-off can be managed using the set size to tune the response time with load. In figures, the hybrid strategy series are labeled "Hybrid-S", where S denotes the set size.

To present realistic response times, we model the inter-node delay with a log-normal distribution with mean 60ms and standard deviation 50ms and total response time as the sum of inter-node delays along a route. The log-normal distribution parameters were selected using results from a study of TCP connection round trip times [3].

We extend our fault model to assume that failed nodes correctly forward lookups

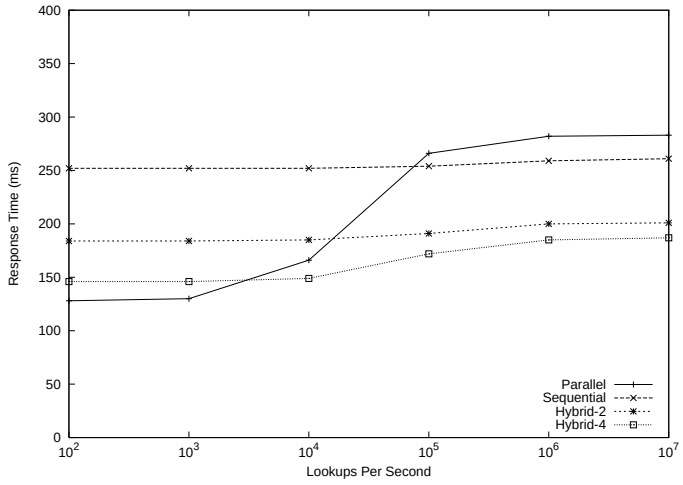
to create added system load, but return incorrect responses that the query node is able to detect. Therefore, a failed route will result in the same system load as a successful route, but will add to the overall response time of the lookup. In a real system where a failure may result in no response at all, it may be necessary to use a timeout for the sequential and hybrid schemes.

To measure the effect of message queuing, we measure response time for lookup rates varying from 1×10^2 to 1×10^7 lookups per second. Since the underlying physical topology is difficult to predict and we are more concerned with the queuing that results from our query strategy, we modeled queuing in the overlay, rather than in the physical network. We assume that each node in the overlay is a leaf node in the underlying physical topology and has a 1 megabit per second link to its gateway router. Furthermore, we assume that the message size is 1 kilobyte, which is consistent with real Pastry implementations.

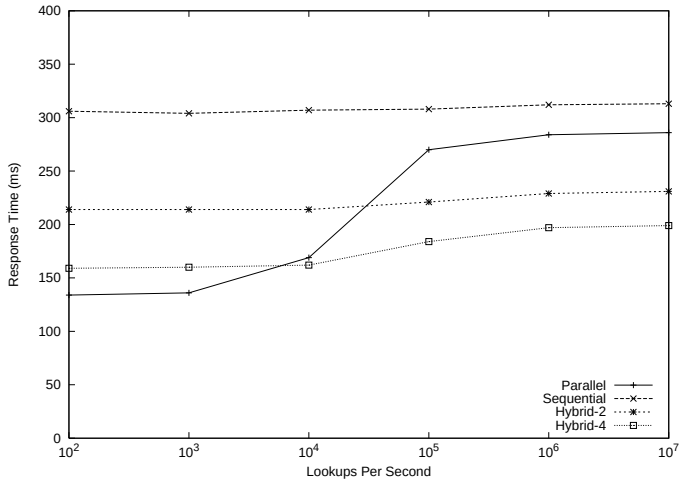
The average response times for the discussed replica query strategies are depicted in Figure 34. We repeated the experiment for f equal to 5%, 10%, and 15%; these results are shown in Figure 34(a), 34(b), and 34(c), respectively.

The trend across these graphs confirms our intuition that the response time of the sequential strategy increases with the fraction of nodes compromised in the network. Furthermore, the response time does not seem to vary significantly with changes in the lookup rate because the effects of an increased lookup rate are not compounded by the parallelization of replica queries.

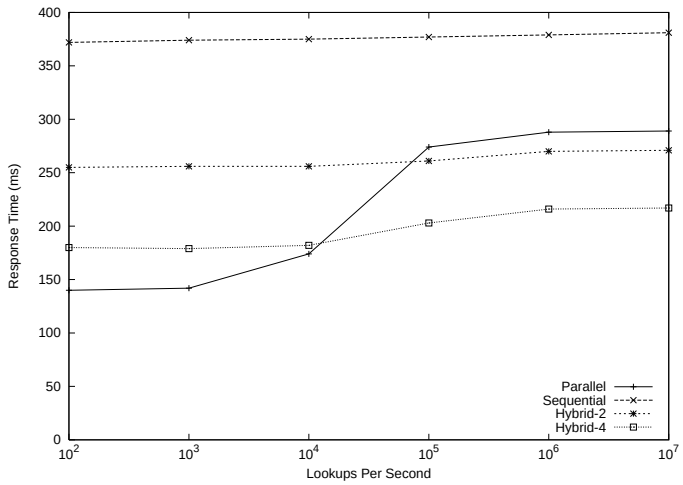
To the contrary, the parallel strategy is not resilient to changes in the lookup rate. As the lookup rate increases beyond 10000 lookups per second, the response time of the parallel strategy increases dramatically. With a relatively low fraction of nodes compromised (Figure 34(a)), the sequential strategy can even outperform the parallel strategy in terms of response time. The additional load resulting from parallelization results in message queuing and delayed responses. This effect may



(a)



(b)



(c)

Figure 34: Response time (ms) for successful lookups with increasing lookup rate with $f =$ (a) 5%, (b) 10%, and (c) 15%. ($N = 2^{28}, n = 8192, B = 16, r = 16$)

be stronger in small networks, which have reduced capacity, and in networks with a higher replication degree.

The hybrid strategies offer a suitable trade-off between the parallel and sequential strategy. The set size can be increased to reduce response times and improve resilience to changes in the fraction of compromised nodes. To improve the resilience to changes in lookup rate, the set size should be decreased. This value of this parameter should be determined by the needs of the application. For example, caller lookup rates in a voice over IP (VoIP) application may change wildly with the time of day and a smaller set size should be used to control congestion.

5.4 Discussion

In this chapter, we characterized a class of DHTs, which employ a tree-based routing scheme. We proved that MaxDisjoint, a replica placement for tree-based routing DHTs, creates disjoint routes. Furthermore, we determined the number of replicas necessary to produce a desired number of disjoint routes. Through simulation, we showed that this placement creates disjoint routes effectively in DHTs that are sparsely populated or populated in clusters. In addition, MaxDisjoint creates disjoint routes without modification of the underlying routing scheme; therefore, its implementation is independent of the underlying DHT chosen, provided that the underlying DHT employs tree-based routing. Furthermore, we demonstrated that disjoint routes have a positive impact on routing robustness and the probability of routing success when nodes are compromised at random or in runs. Specifically, we showed that a vast majority of queries can be resolved successfully even with a quarter of nodes compromised.

We also compared our replica placement with another mechanism for creating route diversity called neighbor set routing. MaxDisjoint has a stronger impact on routing robustness than neighbor set routing; however, when the mechanisms

are combined, substantial benefit is gained, especially with a large fraction of nodes compromised. Therefore, using two or more route diversity mechanisms, like replica placement and neighbor set routing, can have a positive impact on routing robustness.

Finally, we considered some of the practical limitations of using MaxDisjoint in a real implementation; that is, we evaluated the choice of the replica query strategy on response time. Of particular concern was the impact of a parallel strategy on the system load and, as a result, the response time. We observed that the parallel strategy is adversely affected by an increase in the lookup rate; however, it is resilient to changes in the fraction of nodes compromised. Conversely, the sequential strategy is not significantly affected by changes in the lookup rate, but the response time increases with the fraction of nodes compromised. As a solution, we offered a hybrid scheme, in which sets of two or more replicas are queried sequentially. We explained how the set size could be tuned to give a reasonable response time and resilience to changes in the lookup rate.

Before concluding this chapter, we feel it is worth mentioning the so-called "one hop" DHTs [18, 19, 25]. These DHTs attempt to create routes with $O(1)$ hops by restructuring the id space and maintaining more routing state at each node. Since these DHTs have short routes, the likelihood that two routes intersect is dramatically reduced. Therefore, most replica placement schemes can sufficiently create route diversity in these DHTs. However, it comes at the expense of extra routing state at each node that is inherent in the construction of one hop DHTs.

CHAPTER VI

COST-CONSCIOUS DISTRIBUTED HASH TABLE

Although efficient in routing at the logical layer, structured p2p overlays are generally agnostic to costs incurred at the physical layer. Locality-aware protocols like Pastry [32] attempt to minimize costs in the physical layer by selecting neighbors that are nearby according to some locality metric (e.g., round-trip time). Clearly, this is beneficial in reducing latencies for clients of the lookup service, but there are also costs to the underlying network operators.

An Internet service provider (ISP), for example, may incur transit costs to exchange messages with another ISP with which it does not peer. The economic relationships between ISPs play a significant role in Internet routing. Paths in the Internet may be efficient with respect to transit cost, even at the expense of latency.

When an overlay is deployed on the Internet, these transit cost benefits may be lost. Structured p2p overlays, for example, create multihop routes, where each logical hop in the overlay route connects to Internet end hosts. Each of these hops corresponds to a single Internet route, which may traverse many networks. In the worst case, every hop of an overlay route may incur transit costs; this is especially wasteful when the source and destination of the overlay route are clients of the same ISP. In general, overlay paths incur several times the transit costs of the direct Internet path from source to destination.

The so-called "Net Neutrality" debate demonstrates that peer-to-peer technology is making a noticeable economic impact on ISPs. The debate was spurred by Comcast's throttling of Bittorrent traffic [20]. Karagiannis et al. [22] studied locality in

Bittorrent and showed that Bittorrent severely increases an ISP's bandwidth requirements. In fact, they showed that Bittorrent is "locality-unaware"; 70-90% of locally available pieces are downloaded from external peers. Clearly, peer-to-peer overlay designers must be more conscious of the networks that host their services. In particular, overlays should avoid using excessive resources and incurring unnecessary IP transit costs for ISPs.

In this chapter, we consider a scenario wherein several ISPs are cooperating to deliver a DHT service to their clients with minimum shared cost. We create a routing infrastructure that forwards messages using prefix-matching within an organization-based id space. We will show that this scheme ensures that overlay routes incur a bounded transit cost significantly less than traditional DHT implementations. Furthermore, using MaxDisjoint (Chapter5) as an example, we show that replica placements that distribute replicas across the id space can be used to further reduce routing costs while improving object availability. We find the optimal replication degree that balances the storage of replicas with transit costs. We support our analytical findings with detailed experimental results from an Emulab [41] deployment of a lookup service implementation we have developed using Rice University's FreePastry package [1]. Our experiments demonstrate that using our design can achieve a significant reduction in cost ($\sim$80%) compared to a typical Pastry deployment with random node id assignment and neighbor set replication. Furthermore, we discuss how our scheme accounts for varying object popularities and compare its cost performance versus caching.

6.1 System Model

The Internet is the interconnection of numerous networks located worldwide. To the end user, the manner in which the networks are interconnected is of little importance so long as connectivity and quality of service are not compromised. In reality, the

interconnection is a hierarchy whose structure is governed by economics. To be a part of the Internet, a network must connect to at least one other network that is already part of the Internet. Just as the customers of an ISP pay for bandwidth, an ISP may have to pay another network operator for the necessary bandwidth to connect its network to the Internet. The cost for this exchange is called IP transit. In the event that the two networks agree to exchange bandwidth without any monetary compensation, the two networks are said to be peers. These peering and customer-provider relationships create a hierarchy wherein the top tier of networks (Tier 1 ISPs) are able to reach any network in the Internet without incurring any cost.

In this chapter, we consider the scenario where several ISPs wish to share a peer-to-peer distributed hash table with a shared cost. Clearly, this distributed system will require the exchange of information between ISPs and may incur excessive IP transit costs if not designed without the interconnection relationships in mind. We assume that all participating ISPs are connected to the Internet and the transit costs to transmit a message from any participating ISP to another are known. We make no additional assumptions regarding the interconnection of the ISPs, as depicted in Figure 35. Two participating ISPs may be peers (ISPs B and C), a customer and provider (ISP D provides service to ISP F), or connected through third parties that are not participating in the overlay (ISP A is connected to ISP B through ISPs H and J).

Using the known interconnection, we can create a fully-connected logical topology from the participating ISPs as shown in Figure 36. This logical topology is formed as follows. Peering ISPs are combined into a single node (ISPs B and C are combined into node BC). Each of the remaining ISPs becomes a single node. In the remainder of this chapter, we will refer to each node in this logical topology as an organization.

We place an edge between each pair of nodes with a weight that corresponds to the minimum transit cost between the representative ISPs. For example, the weight

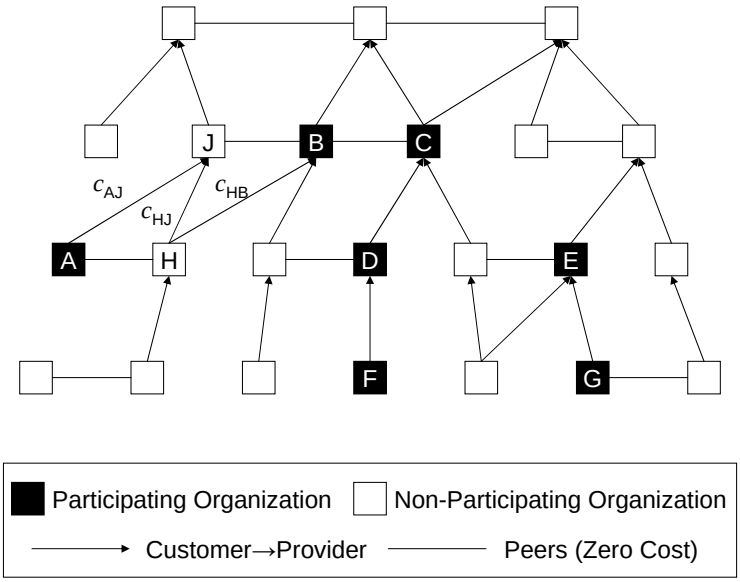


Figure 35: Example of the economic relationships between ISPs.

of the edge between node A and node BC is determined from the minimum of all the possible IP transit costs between ISPs A and B. Note that ISP A can connect to ISP C through ISP B at zero cost. Using this logical topology, each ISP can have its clients maintain a routing table annotated with the costs to route to the other participating ISPs. This table may be used to facilitate cost minimization.

We assume that the ISPs are sharing the operating costs of the overlay and can gain no advantage from misbehavior. We believe such a venture is plausible given that the economic strain of peer-to-peer traffic has already caused ISPs to change their operating practices. A shared overlay gives the ISPs the opportunity to provide the rich services of p2p technology to their customers without incurring excessive cost. In addition, ISPs may be able to develop a revenue model around the shared overlay.

6.2 Distributed Hash Table Design

Our cost-conscious DHT design is comprised of three components that are deployed over any structured overlay that employs prefix-matching routing: a node id assignment scheme, replica placement, and cost minimization function. We discuss these

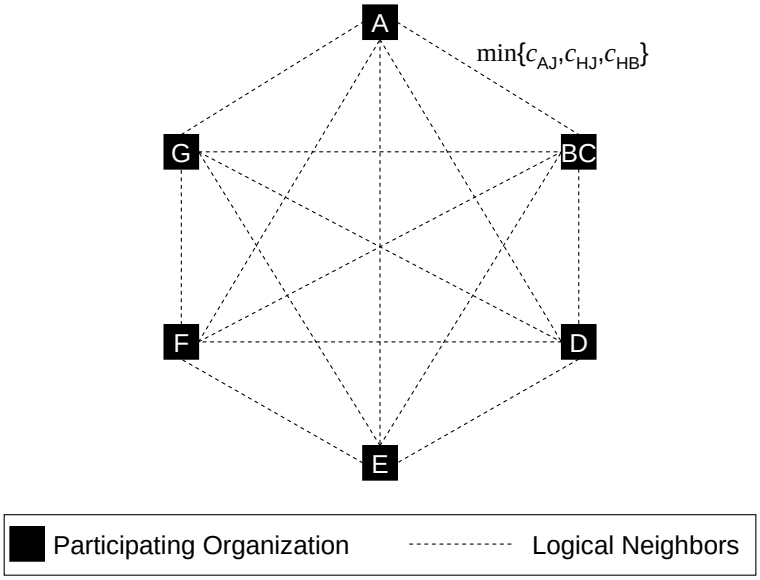


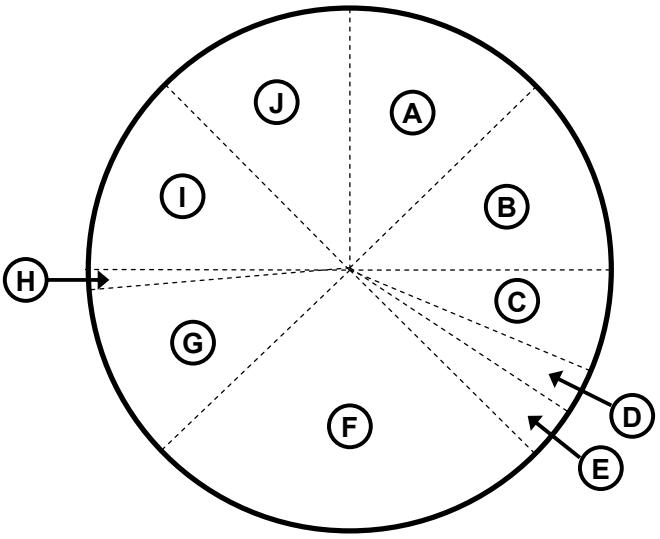
Figure 36: Example logical topology formed between ISPs participating in the cost-conscious overlay.

components in detail in the following sections.

6.2.1 Organization-Based Node Id Assignment

Node ids are assigned using an organization-based technique; that is, node ids within a single organization share a common prefix. The remaining digits are assigned randomly to each node. As such, nodes from the same organization reside in a common segment of the overlay id space. This approach differs from a locality-based id assignment scheme because nodes that are "close" to each other in terms of network locality are not necessarily from the same organization. Similarly, an organization-based scheme differs from a network- or location-based [44] scheme because an organization may be composed of several networks.

The id prefix determines the location and size of the id space segment assigned to an organization. To ensure coverage of the entire id space and balance load, we assign segment sizes roughly proportional to the organization sizes. Note that this can be accomplished simply by varying the prefix length. For example, consider a node id assignment for Pastry with $b = 4$. A prefix of length 1, like `0x5...`,



Organization	Prefixes
A	0x0-0x1
B	0x2-0x3
C	0x4
D	0x50-0x57
E	0x58-0x5F
F	0x6-0x9
G	0xA-0xBB
H	0xBC-0xBF
I	0xC-0xD
J	0xE-0xF

Figure 37: Example of organization-based node id assignment.

corresponds to a segment size of 1/16 ($1/2^b$) of the id space whereas a prefix of length 2, like `0x3F...`, corresponds to a segment size of 1/256 ($1/2^{2b}$) of the id space. Furthermore, an organization can take several adjacent prefixes of different lengths to better approximate its proportion of the id space. This is depicted by the differently sized segments in Figure 37.

With an organization-based id assignment scheme, we claim that overlay paths will incur a bounded transit cost. We state this formally in the following theorem.

Theorem 6. *Consider a full distributed hash table that employs prefix-matching routing and organization-based id assignment. Suppose that the maximum transit cost to route between any pair of organizations is c. The cost of the path from any node a to node x in organization X incurs a transit cost of at most $c \cdot |x|$, where $|x|$ denotes the length of the organization prefix assigned to node x.*

Proof. The intuition behind this theorem is derived from the prefix-matching routing scheme. With each hop in the overlay path, the shared prefix between the next hop and the destination increases by one or more digits. Therefore, once a path enters the destination organization (as defined by its prefix), it will not leave that organization.

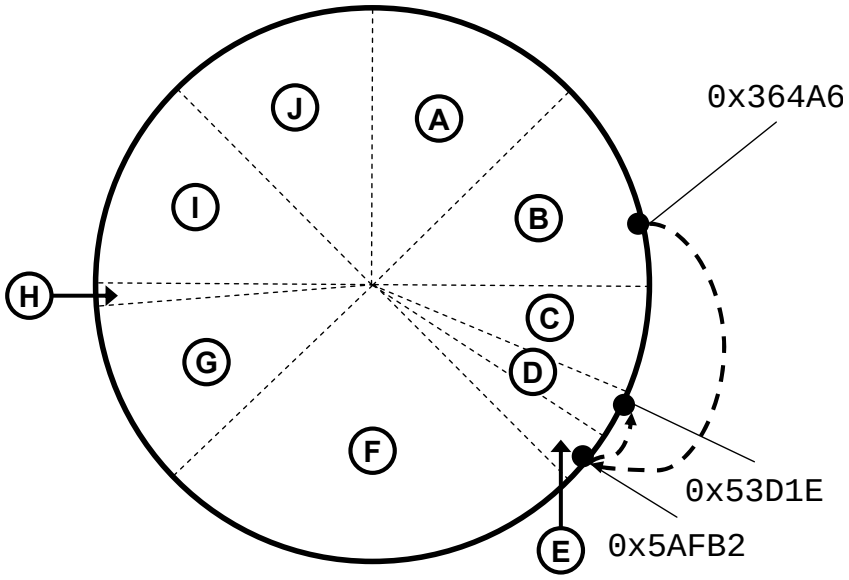


Figure 38: Example of a path using organization-based node id assignment.

Prefix-matching routing ensures that this will occur in at most $|x|$ hops. If we incur the maximum transit cost c with each of the first $|x|$ hops, then the path incurs a cost of $c \cdot |x|$. □

An example path from a node in Organization B to a node in Organization D is depicted in Figure 38. Note that the path traverses two organizational boundaries, which is the length of the prefix assigned to the destination node in Organization D.

6.2.2 Replica Placement

Organization-based id assignment provides a method for bounding the transit cost of a single path. With multiple disjoint paths to choose from, additional benefit in transit cost reduction may be realized. In previous chapters, we showed that replica placement can be used to create disjoint paths.

Replica placements that distribute replicas across the id space can reduce cost in two distinct ways. First, replication increases the likelihood that an organization has a replica of any given object. Since there is zero transit cost within an organization, any lookup for an object that is replicated locally incurs zero cost. For example, in Figure 39, since the source node `0x92B38CD4` and the replica `0x9B82ED82` share a

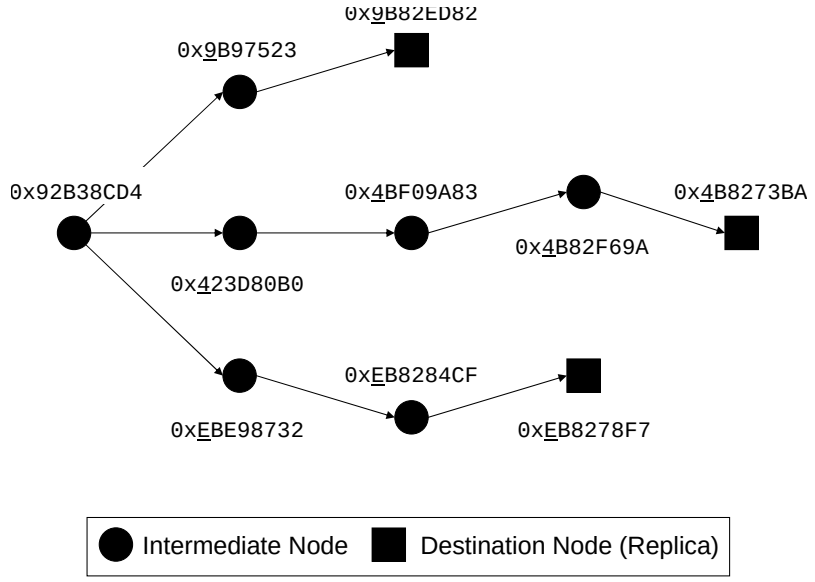


Figure 39: An example of three disjoint routes created using MaxDisjoint replica placement.

common prefix, it is likely that they reside in the same organization and the cost of the path between them is zero.

Second, if there is no local replica, there are multiple disjoint paths to other replicas from which the source node may choose. If each node maintains a table of the costs from its own organization to the others, it may choose the replica that can be fetched with minimal transit cost.

Furthermore, in a real deployment of our DHT, we may encounter node failures, which may impact the ability to route to and fetch an object. The natural solution is to replicate objects to increase the probability that an object is accessible and reachable. If the replica placement creates disjoint paths, then routing robustness may improved as well. Therefore, in the remainder of this work, we will use MaxDisjoint to demonstrate the benefits of replica placements that distribute replicas across the id space.

Suppose we want to replicate objects to the extent that all lookups are resolved with zero transit cost. This requires that each organization replicates the entire object set. We state the replication degree necessary to ensure that a replica exists in each

organization in the following theorem.

Theorem 7. *Consider an overlay that employs a prefix-matching routing scheme, organization-based id assignment, and* MaxDisjoint *replica placement. Let p be the length of the maximum length prefix assigned to any organization. Each organization will have at least one replica of any object that is replicated 2^{bp} times, where b is the prefix-matching base.*

Proof. With 2^{bp} replicas, MaxDisjoint placement ensures that each replica has a unique prefix of length p. Therefore, one or more replicas share a prefix with every organization. □

In addition to the cost benefits, this replication degree ensures that every object can be fetched in the unlikely event that an organization is partitioned from the rest of the overlay. Clearly, replicating objects to this degree is costly and unnecessary in typical deployments of a DHT. Nevertheless, it raises an interesting question: *What replication degree is sufficient to optimize transit and replica storage costs?*

6.2.3 Cost Minimization Function

If the goal of each organization was simply to reduce its own transit costs, then the global replication degree would be chosen as in Theorem 7. The resulting effect would be to replicate the entire object set in each organization. Recall that these replicas must be stored by the clients of each organization. This may be feasible for some organizations, but smaller organizations may find it difficult to expend the necessary resources. Furthermore, as node failures are introduced, the necessary replication degree must be scaled with the failure rate to ensure that each organization has at least one available replica with high probability.

To find the replication degree that best satisfies the needs of all the organizations involved, we define the cost function $C_i(r)$ for organization i that quantifies transit

and storage costs as follows:

$$\begin{aligned} (\text{Total Cost})_i &= (\text{Storage Cost})_i + (\text{Routing Cost})_i \\ C_i(r) &= f_i^{LS}(r) C_i^S(r) + p\left[1 - f_i^{LR}(r)\right] C_i^R(r) \end{aligned}$$

The total cost C_i for an object is expressed in terms of its replication degree r as the sum of the cost incurred for storing objects locally and for resolving queries remotely. The total storage cost is computed from C_i^S, the storage cost per object; and f_i^{LS}, the probability that the object is stored locally. The total routing cost is computed from C_i^R, the routing cost per query routed outside the organization; f_i^{LR}, the probability that the query is resolved locally; and p, the popularity of the object expressed in terms of the number of queries for that object over its lifetime. It is assumed that C_i^S, C_i^R, and C_i are non-negative.

In the following sections, we show how an organization can express its preferences by tuning its cost functions and the impact of those choices on the global replication degree. We omit the subscript i to improve readability, but it is assumed that parameters are tuned on an organization-by-organization basis.

6.2.3.1 Tuning C^S and C^R

An organization can reflect its operational preferences and impact the total cost function by properly tuning the C^S and C^R functions. Typically, C^S will be proportional to the organization's tolerance to locally storing objects. A large organization may have a near zero C^S to reflect its indifference to a high replication degree.

The C^R function reflects the organization's preference to route to replicas stored remotely. Note that the transit cost is dependent on the destination of the lookup. Therefore, C^R must be expressed as the composition of these costs. One method for computing C^R is to take the weighted sum of the transit costs to each organization, where the weights are the probability of routing to that organization. In our initial experimentation, we found that this method of computation was too coarse and did

not yield desirable results. A better approach is to express C^R as a function of r. Indeed, the average transit cost decreases with increased replication degree and this should be reflected in the cost function. Simple simulation can be used to estimate $C^R(r)$.

6.2.3.2 Accounting for Object Popularity

The cost function accounts for object popularity using the parameter p, which expresses popularity as the number of lookups over the object lifetime. With increased popularity, the weight of the routing component of cost is increased and total cost is minimized by increasing the replication degree. Intuitively, the replication degree of a very popular object should be higher than that of one that is rarely accessed.

Alternatively, we can view the cost function from the point of view of amortization. The storage cost is amortized over the lifetime of the object. Therefore, if the object is looked up more frequently than others in its lifetime, we can afford to make a larger investment in its storage cost. Clearly, the object lifetime is a matter of interest. If objects are popular over a very long lifetime, then it is justified to replicate those objects to the extent that there is a replica in each organization. Domain name system (DNS) records are an example of this type of object. The usefulness of the cost function comes with objects that have a relatively short lifetime or objects whose popularity decays over time. The Bass model [5] of demand growth for innovations can be used to model objects whose popularities decay over time. The model follows the behavior of typical viral growth, a logistic curve. This behavior has been observed in video popularity on YouTube [9] and in content distribution networks [15].

Determining object popularity at lookup and insertion is another consideration. For the purposes of this work, we assume that there is a service that determines object popularity over a short period of time starting at the time of insertion. Objects are initially inserted with average popularity and promoted or demoted to the appropriate

Input: key: the key to look up
Data: R: Replication degrees for all popularities, where R_p gives replication degree for popularity p

```
used ← ∅
for r ← R_{max p} to R_{min p} do              /* iterate down popularities */
    Locs ← ReplicaLocations(key, r)
    trgt ← Locs_i s.t. Cost(Locs_i) = min(Cost(Locs)) && Locs_i ∉ used
    res ← Lookup(trgt)                          /* try the minimum cost replica */
    used ← used ⋃ trgt
    if res is not null then exit;              /* miss?  continue :  else, stop */
end
Lookup(key)                                     /* last resort:  try master key */
```

Figure 40: Replica lookup ordering algorithm.

level of popularity as time passes.

Rather than maintaining a mapping of objects to their popularities, we utilize the recursive nature of MaxDisjoint placement; that is, increasing the replication degree simply adds new replica locations without changing any existing replica locations. Therefore, when performing a lookup, we compute all possible replica locations using the replication degree of the most popular object. The actual replica locations are a subset of the computed replica locations regardless of the object popularity. In the process of looking up replicas, some "misses" may occur when a location that does not store the replica is queried. These misses are more likely to occur for less popular objects.

Blindly querying the possible replica locations is not effective in minimizing the total routing cost. Simply querying the replica locations in order of decreasing transit cost does not effectively avoid misses. Therefore, we propose the algorithm in Figure 40 to avoid misses while minimizing routing cost.

The algorithm begins by assuming the object being looked up is one of the most popular objects. The replica locations for this replication degree (R_{maxp}) are computed and the minimum cost replica is queried. If the result is null, then we assume

that we have a miss[1]. In other words, we assume that the object has the next lower level of popularity and the process is repeated.

The algorithm repeats until a non-null result is received. After trying all possible popularities, if we have not received a non-null result, then we query the master key. If the object exists, it must be accessible at the master key.

6.2.3.3 Minimizing Global Costs

To minimize global costs, we attempt to minimize the cost functions of all the organizations in concert. Consider k organizations each with $N_1, N_2, \ldots, N_k$ nodes such that $N = \sum_i N_i$. In practice, each organization has a known customer base of size C_i and can estimate its size N_i probabilistically. Assuming that a customer participates in the overlay with probability p and C_i is sufficiently large, we can estimate $N_i = pC_i$.

Suppose each organization in the system has its own cost function $C_i(r)$. If the replica placement distributes objects uniformly across the id space, we can approximate f_i^{LS} as the fraction of the id space owned by organization i times the replication degree and rewrite $C_i(r)$ as follows:

$$C_i(r) = \left(\frac{Ni}{N}\right) rC_i^S(r) + p[1 - f_i^{LR}(r)]C_i^R(r)$$

Furthermore, f_i^{LR} can be written as a function of N_i, N, and r:

$$f_i^{LR} = \begin{cases} \frac{N_i}{N} r, & r < \frac{N}{N_i} \\ 1, & \text{otherwise} \end{cases}$$

To minimize the total system cost, we find the integral replication degree r that minimizes:

$$C(r) = \sum_i C_i(r)$$

[1] A null result could also be the result of a node failure, but the algorithm assumes no node failures.

It is sufficient to minimize the sum of the organizational costs because all the costs are non-negative functions of r. However, it is necessary to ensure that the organizations assign the cost functions C_i^L and C_i^R using a common metric. Furthermore, since all the necessary parameters are known a priori, the replication degree can be computed in advance.

In some cases, organizations may not want to reveal their cost functions to the others. The cost function can reveal some sensitive data, such as the ISP's storage and transit costs. With some analysis, the nature of these costs can be determined. As an alternative to revealing this information, each organization can propose a replication degree r_i that minimizes its own cost function $C_i(r)$. Given the set of values $R = \{r_i\}$, a global replication degree r must be chosen. The following are some strategies for selecting the global replication degree:

Average Replication Degree A basic strategy is to take the mean of R as the global replication degree. This will create sufficient local replicas for some organizations, but will perform poorly for others. In addition to inconsistent performance across organizations, this strategy is not robust; one organization could easily inflate the average replication degree.

Maximum Replication Degree Another strategy is to select $r = \max R$. This strategy is favorable to organizations that incur a significantly higher cost for resolving queries remotely than locally.

Maximum Non-Outlier Replication Degree Consider the case when there is one organization (or a small number of organizations) that is trying to artificially influence the global replication degree. This organization may produce a replication degree from which it may derive relatively little benefit compared to the cost imposed on the other organizations. In this case, we may omit this outlier when computing

the maximum of R. In other words, we may take the second (or i-th) greatest value in R.

6.3 Experiments

We evaluate our DHT design using a lookup service implementation we developed with FreePastry [1]. We implemented organization-based id assignment and MaxDisjoint placement over Pastry and evaluated it in Emulab.

6.3.1 Experimental Setup

We deployed our lookup service on 32 Emulab nodes. Each node was assigned to one of eight organizations. The organization sizes and transit costs between organizations were assigned at random as shown in Table 6.3.1. We would like to have used actual transit costs, but these values are not publicly available. Nonetheless, we believe that using random costs is sufficient in characterizing the effectiveness of our solution.

To ease interpretation of results, after generating organization sizes at random, we numbered the organizations from largest to smallest. Thirty-two virtual nodes were run on each physical node for a total of 1024 nodes in the overlay. We inserted 5000 objects and performed 10000 lookups, each for a random object originating at a random query node. Each data point in our results reflects the cost incurred for the lookups performed.

6.3.2 Experimental Results

We performed three series of experiments. Before we consider the more realistic scenario of varying object popularities, we evaluate the impact of the choice of C^S and C^R on system cost for uniform object popularity. This experimental setup helps isolate the behavior of our solution to verify its correctness. In the remaining experiments, we vary the object popularity with a more realistic Zipfian distribution. The second compares the cost benefits of our approach with traditional Pastry, and the

Table 3: Cost-conscious DHT experimental setup: (a) organization-based id assignment and (b) inter-organization transit costs.

(a)

Organization	# Nodes	Prefixes
O1	455	`0x0-0x6`
O2	175	`0x7-0x9`
O3	120	`0xA-0xB`
O4	99	`0xC`
O5	84	`0xD`
O6	54	`0xE`
O7	23	`0xF0-0xF7`
O8	14	`0xF8-0xFF`

(b)

	To							
From	O1	O2	O3	O4	O5	O6	O7	O8
O1	-	.20	.92	.37	.04	.99	.36	.48
O2	.68	-	.86	.14	.37	.98	.24	.97
O3	.15	.42	-	.77	.94	.77	.44	.30
O4	.10	.37	.83	-	.32	.40	.49	.72
O5	.47	.95	.84	.90	-	.37	.69	.67
O6	.58	.40	.75	.55	.51	-	.62	.24
O7	.13	.92	.93	.18	.73	.83	-	.94
O8	.04	.83	.91	.79	.91	.35	.88	-

Table 4: Analytically-determined cost-conscious replication degrees.

Case	r_{optimal}	r_{avg}	r_{max}	$r_{\text{max-outliers}}$
Case 1: $C^S = 0.01C^R$	16	16.1	32	16
Case 2: $C^S = 0.1C^R$	5	6.8	16	8
Case 3: $C^S = C^R$	1	1.4	2	2

last examines the system behavior compared to caching. We discuss the parameters and results of these experiments in the remainder of this section.

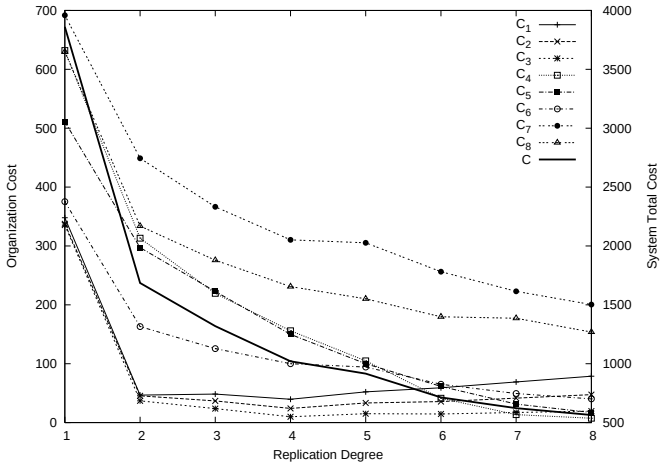
6.3.2.1 Impact of C^S and C^R

We executed lookups of 10000 objects of equal popularity (i.e., each object is looked up with equal probability) and maintained the routing cost at each node in the overlay. Each data point in the organization cost curves represents the sum of the number of objects stored and the transit costs for each organization. We compute the cost for the i-th organization in the system as:

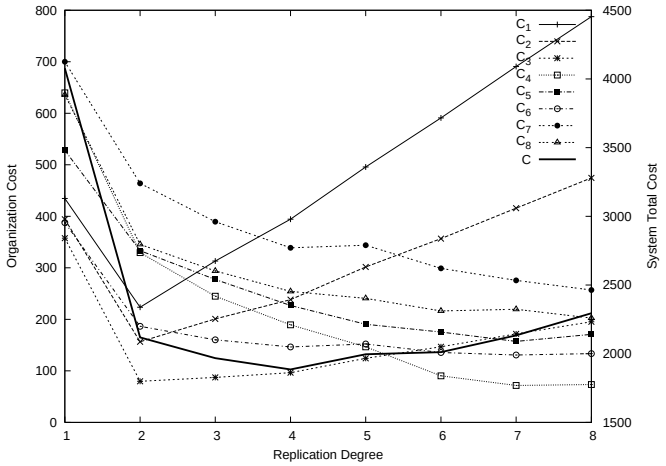
$$C_i = C^S \cdot (\text{objects stored}) + (\text{total routing cost})$$

and the system total cost as $C = \sum C_i$. In Figure 41, we present the results with three values of C^S, referred to as Cases 1, 2, and 3, respectively. Table 4 shows the replication degree that would result from the replication degree selection strategies discussed in Section 6.2.3.3.

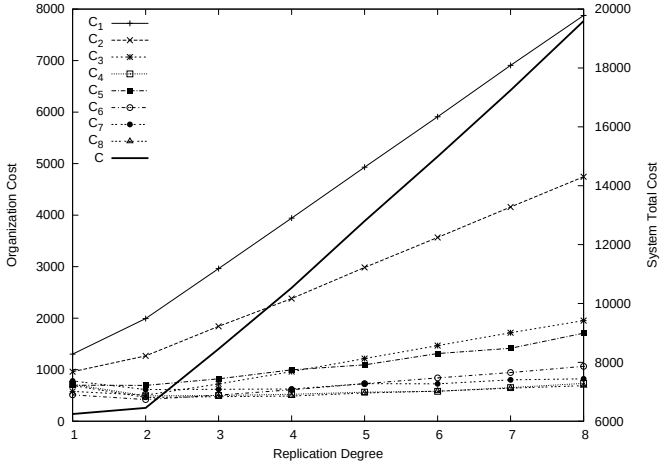
Looking at each case individually, we observe the impact of tuning C^S. For Case 1 (Figure 41(a)) when C^S is negligible relative to C^R, the benefits of having local replicas greatly outweigh the cost of maintaining those replicas. This is similar to the case of an extremely popular object. Therefore, objects should be replicated with a high replication degree in this regime. Our cost function predicts a replication degree of 16, but in practice, we may choose the replication degree prescribed by Theorem 7 (32) and scale it by the failure probability of nodes to ensure that each organization has a live replica with high probability.



(a) Case 1: $C^S = 0.01C^R$



(b) Case 2: $C^S = 0.1C^R$



(c) Case 3: $C^S = C^R$

Figure 41: Organization and system total cost (solid bold line) with increasing replication degree.

In terms of the impact on the individual organizations, we observe that the smaller organizations (C_7 and C_8) are carrying the larger burden of cost. This is due to the fact that a very large replication degree is necessary to mitigate the transit costs incurred by these small organizations.

In Case 2 (Figure 41(b)), C^S has reached the point where there is a replication degree that optimally trades off the storage and remote retrieval costs for resolving a lookup. This is the case where our cost function is most interesting and useful. Our model prescribes a replication degree of 5 whereas the optimal replication degree based on the experimental results is 4. Nonetheless, choosing a replication degree of 5 results in only a 6% increase in total cost over the optimal choice.

Since none of the individual organization costs are minimal at the optimum, it is non-trivial to compute the system optimum without considering all the costs in concert. This is reflected in the alternative strategies, which choose replication degrees that are unnecessarily high to mitigate system cost. Note that in this case the organizations share the burden of cost relatively fairly at the optimum.

When C^S dominates C^R, as in Case 3 (Figure 41(c)), the tendency is to eliminate replication altogether and resort to the remote resolution of queries. This is similar to the case of an unpopular object. In this regime, the cost burden is placed primarily on the larger organizations (C_1 and C_2) because they are responsible for the storage of more objects.

Of the three regimes, we argue that Case 2 is most likely in practice. First, it is unlikely that an ISP would value storage cost as much as routing cost because the storage cost C^S is not a direct cost to the ISP. It captures the more subtle cost of aggravating customers with the storage of an excessive number of objects. This makes Case 3 an unlikely scenario in practice. In Case 1, each organization stores the entire object space. If we assume no failures, then the organizations do not need to rely on each other to resolve lookups. This seems plausible, but it completely ignores the

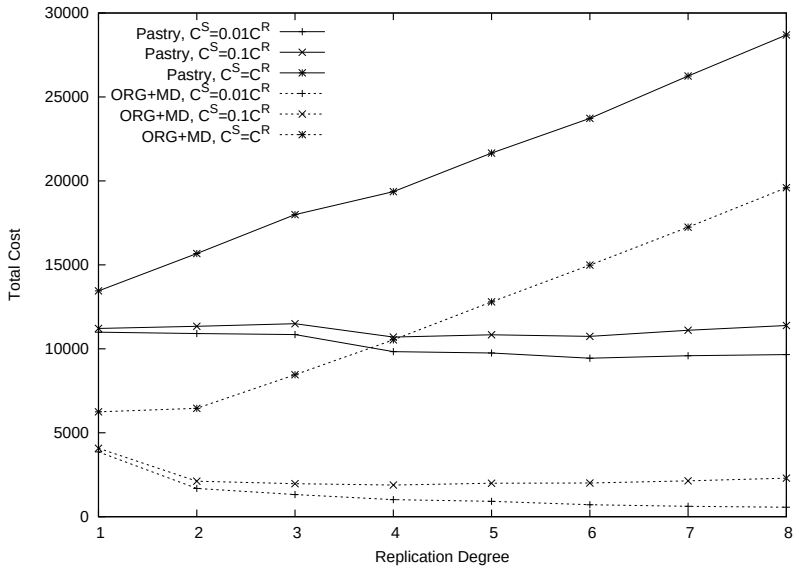


Figure 42: Total cost comparison of organization-based ids with MaxDisjoint (ORG+MD) against Pastry.

effect that the storage of objects may have on the clients of a small organization. In practice, it is unlikely that each organization would be willing to replicate the entire object space. Although we cannot definitively define the relationship between C^S and C^R, we believe that it falls in the regime of Case 2, where our analysis is most useful.

Of the replication degree selection strategies considered, computing the average of all the independently computed replication degrees is the most effective. This strategy best estimates the optimal replication degree provided that all organizations are cooperative and none act selfishly.

6.3.2.2 Comparison with Pastry

To demonstrate the extent of the cost savings of our approach, we compared with traditional Pastry, node ids are assigned randomly and neighbor set replication is used. The total costs of Pastry and our approach (ORG+MD) are shown with increasing replication degree in Figure 42. We omit the organizational costs for clarity.

Comparing the optimal replication degree for each approach, we find overall cost savings of approximately 95%, 80% and 55% for $C^S = 0.01C^R$, $0.1C^R$ and C^R,

respectively. Examining these savings further, we find that organizational-based id assignment alone provides a 55-65% cost reduction (where the replication degree is one). When the storage cost is high, MaxDisjoint provides little benefit because replication is unnecessary. However, the addition of MaxDisjoint increases the savings by as much as 85% when the storage cost is negligible relative to the routing cost.

There is a small reduction in cost with increasing replication degree in Pastry when the storage cost is sufficiently small (see Figure 42, Pastry, $C^S = 0.01C^R$). Regardless of the placement, a larger replication degree increases the likelihood that the organization of the query node will have a replica of the lookup target. However, this cost saving is negligible compared to the transit costs incurred by lookups that must be routed to an external target. Our approach provides an excellent alternative to Pastry when operating cost is a primary concern.

6.3.2.3 Comparison with Caching

Intuitively, caching seems like a natural comparison to our approach. Caching aims to reduce lookup latency by storing copies of the lookup target at each of the intermediate overlay hops along the lookup route. We distinguish this form of caching from one that may cache objects at non-overlay nodes along the Internet path. This alternative form of caching requires that non-overlay nodes have detailed knowledge of the overlay protocol. Caching can be effective in significantly reducing the length of the lookup paths to an object, if the object is cached at a sufficiently large number of overlay nodes. As a result of shortened lookup paths, we would expect a reduction in operating cost.

It is difficult to compare caching to our approach fairly; therefore, we erred on the side of caution by employing cache sizes large enough to store the entire object set. Therefore, none of the cached objects are evicted during the experiment. We would

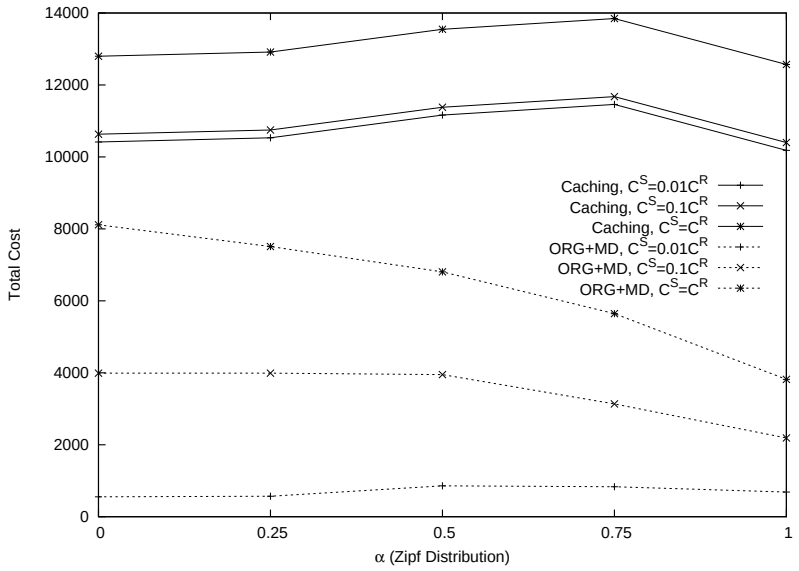


Figure 43: Total cost comparison of organization-based ids with MaxDisjoint (ORG+MD) against caching.

expect this configuration to give the best case performance of caching.

Previous work has shown that Web object requests follow a Zipf-like distribution [6]. In a Zipfian distribution, the frequency of an object is inversely proportional to its rank. Therefore, objects with the highest rank are looked up most frequently. The distribution is characterized by a single parameter α that is used to define the frequency of an object of rank k in a set of size N to be:

$$f(k, N, \alpha) = \frac{1}{k^s H_{N,\alpha}},$$

where $H_{N,\alpha}$ is the N-th generalized harmonic number. When $\alpha = 0$, then all objects have uniform popularity. As α increases, the popularity of the highest ranked objects increases. We compare caching with our approach for a range of α values from zero to one in Figure 43.

From our experimental results, we find that caching is not as effective as our approach in reducing costs. Although caching creates copies of popular objects throughout the network, there is no way for a node to determine the locations of the cached copies. To achieve a noticeable reduction in cost, an object must be cached at nearly

every node. The number of lookups necessary to reach this level of caching is quite large, which explains why we do not see a reduction in cost for caching until $\alpha = 1$.

To the contrary, our approach replicates objects at known locations in the network that can be determined by every node. Therefore, a node can compute the replica locations of the target and choose the replica that can be fetched with minimal cost. The resulting effect is a significant reduction in cost that increases with α. The increasing cost reduction with α can be explained in two ways. First, with a smaller set of highly popular objects, the storage cost investment is quickly amortized by the routing cost savings. These significant reductions in routing cost make a dramatic impact on the total cost. Second, as the set of highly popular objects shrinks, the set of unpopular objects grows. These unpopular objects can be replicated to a lesser degree, which corresponds to a reduction in storage cost.

The results in Figure 43 assume that the popularity of the lookup target is known. If the popularity is known, then the replication degree can be computed and the lookup misses discussed in Section 6.2.3.2 can be avoided altogether. However, in practice, it may be difficult to determine the object popularity at lookup time. Therefore, we measured the performance of the technique described in Figure 40. The results are shown in Figure 44.

The results show that the replica ordering algorithm prescribed in Figure 40 does not work as effectively as one would hope. Nevertheless, the results do help to explain why the technique does not work and how it can be improved. In general, we see an increase in cost followed by a slow reduction in cost with increasing α. The increase in cost can be explained by the large number of misses that occur when the frequency distribution of lookups is near-uniform. The reduction in cost begins when the lookups are concentrated on the most popular objects. Since misses do not occur for the most popular objects, the benefits of replication are more pronounced.

Intuitively, the replica ordering algorithm is a linear search for the target's true

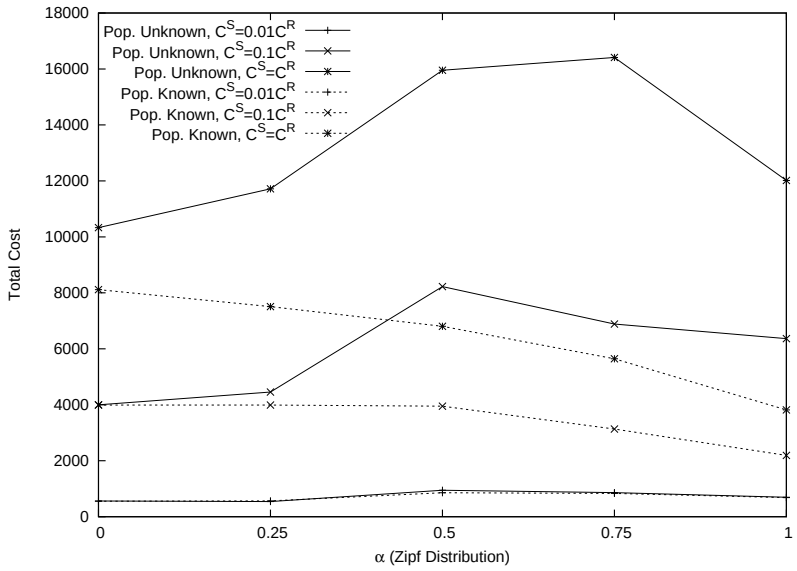


Figure 44: Impact of known (Pop. Known) and unknown (Pop. Unknown) object popularity on the cost of organization-based ids with MaxDisjoint.

popularity beginning with the highest level of popularity. As far as search algorithms are concerned, a linear search is one of the least efficient. However, when it is more likely than the target has the highest level of popularity (which is the case when α is large), a linear search is quite effective. When α is small, it may be more effective to use another search algorithm, a binary search, for instance.

6.3.2.4 Behavior with a Selfish Organization

To this point, we have assumed that the organizations cooperate to provide a DHT service to their users at a minimum cost. Although we assume that organizations are cooperative and share the total system cost (which should discourage selfishness), in reality, ISPs are competing entities and there may be incentives for selfishness. Interesting questions to ask are *To what extent can a "selfish" organization manipulate the DHT operation to its own benefit?* and *How much impact will such manipulation have on the costs of other organizations and the total system cost?*. In the remainder of this section, we report on experiments performed to address these questions.

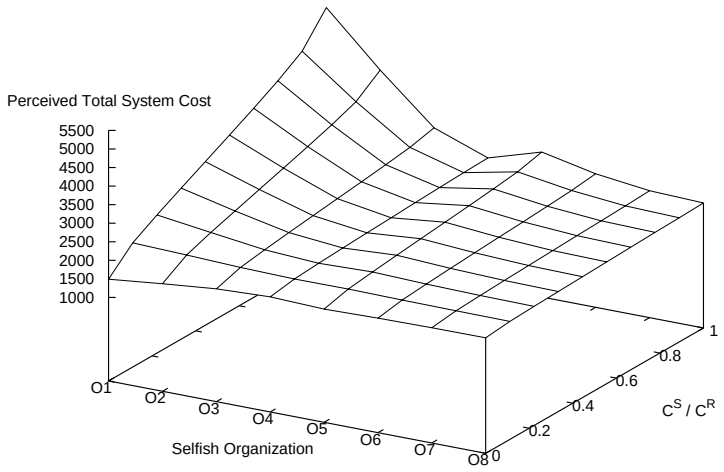


Figure 45: Perceived total cost for a single selfish organization ($r = 4$).

We assume that all organizations set $C^S = 0.1C^R$ except for one selfish organization. We let each organization act selfishly one at a time and vary C^S for the selfish organization from 0 to C^R. The resulting perceived total cost for $r = 4$ is shown in Figure 45.

The plot in Figure 45 is representative of the behavior across all values of r; that is, the largest organizations can significantly affect the perceived cost by simply changing C^S. A selfish organization has two basic strategies it can use to skew the perceived total cost curve and influence the replication degree. One, the selfish organization can increase the perceived cost of the true optimal replication degree. Alternatively, it can decrease the perceived cost of its desired replication degree. We believe these strategies are essentially symmetric, therefore, we consider only the former.

Referring to cost curve C_1 in Figure 41(b), we see that Organization 1 (O1) has an opportunity to reduce its cost by using a replication degree of $r = 2$ or $r = 3$. O1 can force these replication degrees by using $C^S = C^R$ or $C^S = 0.2C^R$, respectively. As a result of this skewed perceived cost curve and the forced replication degree, the selfish organization will benefit and other organizations (and the system) may suffer.

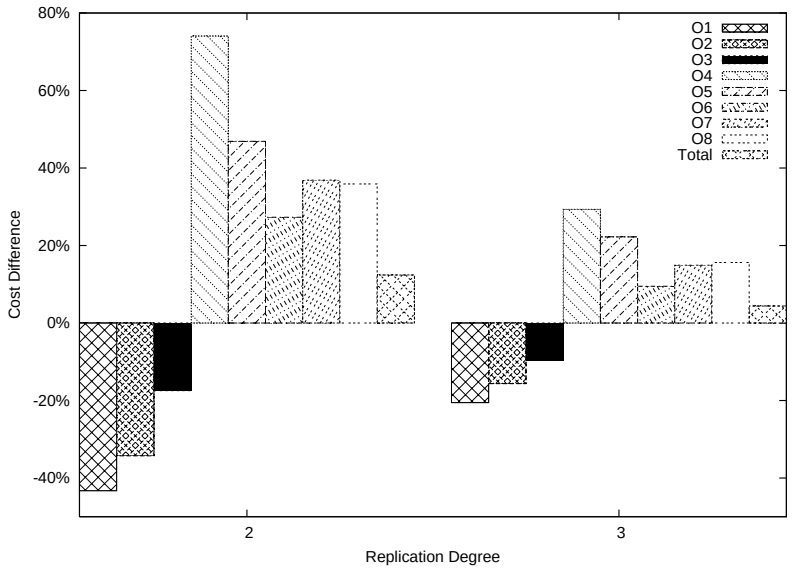


Figure 46: System and organization cost difference using a replication degree other than 4.

We computed the difference in actual system cost and the costs for each organization with these replication degrees as shown in Figure 46.

If O1 is able to change the replication degree to 3, it will reduce its own costs by 21%. Interestingly, this only results in a 4% increase in the total system cost. Of course, this comes at the expense of other organizations, O4 in particular. Similarly, if O1 is able to reduce the replication degree to 2, its costs will drop by 47% while increasing the total system cost by 12%. In this case, the cost of O4 is increased by almost 80%. Furthermore, the largest organizations all benefit by reducing the replication degree. Therefore, these organizations may collude to reduce their costs without raising suspicion among the smaller organizations.

6.4 Discussion

We have presented a distributed hash table design for Internet service providers that is conscious of the costs incurred by inter-organization query resolution and from storing replicated objects. At the heart of the design is an organization-based id assignment scheme that ensures that overlay paths traverse a bounded number of organizational

boundaries. Transit costs are reduced and object availability is improved by using a replica placement, like MaxDisjoint, that distributes the replica set across the id space. We provide a cost function that can be used to minimize costs within the operational parameters of the system. Alternatively, we offer three replication degree selection strategies that can be used to set the replication degree in a distributed fashion without revealing individual organization costs.

Experimentally, we showed that our approach achieves a significant reduction ($\sim$80%) in cost compared to a typical Pastry deployment. Furthermore, our technique outperforms other promising techniques, like caching. We hope that the cost savings we have presented provide an incentive for ISPs to become more accepting of the p2p movement, which has the potential to deliver rich services more efficiently than client-server solutions.

CHAPTER VII

CONCLUSION AND FUTURE WORK

Although many applications have realized the efficiency, scalability, and resilience that peer-to-peer technology can provide, there is a still significant amount of skepticism surrounding p2p. Although the distributed nature of a p2p solution provides some resilience, the level of dependability and fault tolerance that can be designed into critical systems deployed over p2p is still in question. Furthermore, network operators are troubled with the costs of p2p applications deployed on their networks because p2p is generally agnostic to the transit costs associated with Internet routing. Before p2p can become pervasive in practice, these concerns must be addressed and this dissertation has begun an investigation into these problems and potential solutions.

Before we detail the contributions of this dissertation, we would like to state what we feel are the fundamental conceptual findings of our work:

1. Route diversity (disjoint routes, in particular) has a noticeable, positive impact on routing robustness and, in turn, the probability of lookup success in structured peer-to-peer overlays.

2. Replica placement is an effective method for providing route diversity while improving object availability. There exists a replica placement that creates disjoint routes in any distributed hash table that employs tree-based routing.

3. In addition to improving routing robustness and object availability, replica placement can be used with organization-based id assignment to create a distributed hash table implementation that is conscious of the costs to the underlying network operators.

These concepts are the heart of this work and this dissertation provides the analytical and experimental evidence to support these claims.

7.1 Dissertation Summary

In support of the claims above, this dissertation has made the following contributions:

- We proved that equally-spaced replica placement creates disjoint routes in Chord. Furthermore, we showed that d disjoint routes can be created with 2^{d-1} equally-spaced replicas. (Chapter 4)

- Through simulation, we evaluated equally-spaced placement in Chord and showed that it is more effective in creating disjoint routes than a neighbor set or random placement. This benefit is maintained in sparsely populated and clustered id spaces. (Chapter 4)

- The increased number of disjoint routes parlays into an increase in the probability of lookup success and routing robustness. With only four replicas and a quarter of the id space compromised randomly or in runs, simulation results indicated that lookups are successfully resolved 98% of the time. (Chapter 4)

- We defined a class of distributed hash tables that employ a scheme we call tree-based routing. We proved a number of properties for DHTs of this type. (Chapter 5)

- We proposed MaxDisjoint, a replica placement that creates disjoint routes in tree-based routing DHTs. We formally proved that it creates disjoint routes. Furthermore, we determined the replication degree necessary to create a desired number of disjoint routes. (Chapter 5)

- We showed that MaxDisjoint and equally-spaced placement both utilize the properties of tree-based routing, but MaxDisjoint is a more adaptive and

flexible placement in prefix-matching DHTs. (Chapter 5)

- Through simulation, we showed that MaxDisjoint, like its sister equally-spaced placement, creates disjoint routes more effectively than other placements. Again, this benefit is maintained in sparsely populated and clustered id spaces. (Chapter 5)

- MaxDisjoint also improves routing robustness and the probability of lookup success. Simulation results indicated that lookups are resolved successfully 97% of the time with eight replicas and a quarter of the id space compromised randomly or in runs. (Chapter 5)

- We showed how the benefits of replica placement can be enhanced with another route diversity technique we call neighbor set routing. Although neighbor set routing does not create disjoint routes, it does improve routing robustness by creating some additional diverse paths. When neighbor set routing and MaxDisjoint replica placement are combined in Pastry with half of its nodes compromised at random, the probability of lookup success increases to 84% from 52% with replica placement alone. (Chapter 5)

- We evaluated how the implementation of replica set lookup affects system load and response time. We found that querying the entire replica set in parallel may induce unnecessary load in the system, which reduces lookup time. We proposed a hybrid query strategy that trades off system load and response time as appropriate for the operating environment. (Chapter 5)

- We outlined a model for grouping Internet service providers into organizations such that the intra-organization transit costs are zero and the inter-organization transit costs are bounded. Using this model, we proposed an organization-based id assignment and proved that prefix-matching routes in the id space

have bounded transit cost. (Chapter 6)

- We apply MaxDisjoint in this context to create disjoint routes, each with a bounded cost. This provides two opportunities for reducing cost: (1) with multiple replicas, there may be a replica that can be fetched within the organization at zero cost, and (2) with multiple disjoint routes each with bounded transit cost and no local replica, the minimum cost route may be selected. (Chapter 6)

- Because replication has a non-trivial cost, we provide a cost function that may be used to find the replication degree that optimally trades off the cost of replication and transit costs. (Chapter 6)

- We developed and implemented organization-based id assignment and MaxDisjoint replica placement in an open source Pastry distribution from Rice University called FreePastry. We evaluated it using the Emulab facility. (Chapter 6)

- We evaluate the cost of our approach compared to a traditional Pastry deployment with neighbor set replication. Our approach achieves an 80% reduction in cost over Pastry with neighbor set replication in typical operating conditions. (Chapter 6)

- We compare the cost of our approach compared to a traditional Pastry deployment with caching. Although caching is more effective than neighbor set replication, our approach still achieves a significant reduction in cost (60-70%). (Chapter 6)

7.2 Future Work

Our work has opened the door on several avenues for future work. First, we will discuss specific enhancements that could be made to the solutions in this dissertation. Finally, we will conclude with a discussion of the general areas for future work.

The following enhancements to our work serve primarily to weaken assumptions that we have made:

- Our replica placement assumes self-verifying data such that a client can verify the integrity of the response. Alternatively, we could develop a Byzantine fault tolerant algorithm to determine the correct response from a set of untrusted responses. Intuitively, if the source node queries only d replicas that are retrieved along d disjoint routes and we bound the number of faulty routes, then we can design an election protocol.

- Our replica placement assumes that objects are located a specific locations in the id space, but does not specify how replicas are initially placed at those locations. Alternatively, we could design a fault tolerant replica dissemination protocol. We believe that our work with neighbor set routing could serve as first step in the solution because neighbor set routing creates some diversity to a single location. If objects are rarely inserted, we may be able to incur more overhead and use an alternative routing structure to disseminate replicas. We may consider hypercubes because an n-degree hypercube has n node-disjoint paths between every pair of nodes. Therefore, we could construct disjoint paths from the dissemination root to each replica location and increase the routing robustness.

- Our hybrid query strategy is sufficient when the traffic model is known and fairly static. However, a more appropriate solution would dynamically change the degree of parallelization to reduce query load when congestion is detected. This solution may share some characteristics with TCP congestion control.

- Our cost-conscious distributed hash table assumes that object popularity is determined by a dedicated service that can determine popularities over a small number of lookups. Determining the nature of this service is an area for future

work. It may be possible to estimate the lookup rate and approximate the parameters of the Bass model curve to determine the object popularity.

- Our cost-conscious distributed hash table assumes that the participating organizations are cooperative. Although a shared overlay is a joint venture among several Internet service providers, the ISPs are competing entities and there may be an incentive for selfish behavior. Designing a cost-conscious distributed hash table that is robust to selfishness is a topic for future work.

The following are general areas for future work:

- We have demonstrated how replica placement can create route diversity in tree-based routing distributed hash tables. However, we believe there are other techniques we may use to create route diversity in structured p2p overlays.

- Our cost-conscious distributed hash table demonstrates how design can be used to maintain the benefits of p2p while operating in a manner that is more conscious of network operators. Although it is a significant step forward, it is far from ideal. Finding a suitable solution to this problem is critical to the livelihood of p2p technology.

- Although we have not discussed it at any length in this dissertation, it is interesting to note that many of the functions of the network layer are reproduced in a p2p overlay, e.g., addressing, routing, end-to-end reliability. Rather than replicating this functionality, perhaps the network layer should be more receptive to the needs of the applications of this day and age. Obviously, an Internet redesign is no easy task (and may come with much opposition), but it is an open problem that deserves investigation. Researchers with knowledge of p2p technology should play a very active role in this research.

REFERENCES

[1] "FreePastry." http://freepastry.rice.edu/FreePastry/. Last accessed: August 2008.

[2] Adya, A., Bolosky, W. J., Castro, M., Cermak, G., Chaiken, R., Douceur, J. R., Howell, J., Lorch, J. R., Theimer, M., and Wattenhofer, R. P., "Farsite: federated, available, and reliable storage for an incompletely trusted environment," in *Proceedings of OSDI '02*, pp. 1–14, 2002.

[3] Aikat, J., Kaur, J., Smith, F. D., and Jeffay, K., "Variability in TCP Round-Trip Times," in *Proceedings of ACM SIGCOMM IMC'03*, pp. 279–284, 2003.

[4] Artigas, M. S., Lopez, P. G., and Skarmeta, A. F. G., "A Novel Methodology for Constructing Secure Multipath Overlays," *IEEE Internet Computing*, vol. 9, no. 6, pp. 50–57, 2005.

[5] Bass, F. M., "A New Product Growth Model for Consumer Durables," *Management Science*, vol. 15, no. 5, pp. 215–227, 1969.

[6] Breslau, L., Cao, P., Fan, L., Phillips, G., and Shenker, S., "On the Implications of Zipf's Law for Web Caching," in *Proceedings of INFOCOM'99*, pp. 126–134, 1999.

[7] Castro, M., Druschel, P., Ganesh, A., Rowstron, A., and Wallach, D., "Secure Routing for Structured Peer-to-Peer Overlay Networks," in *Proceedings of OSDI'02*, pp. 299–314, 2002.

[8] Castro, M., Druschel, P., marie Kermarrec, A., Nandi, A., Rowstron, A., and Singh, A., "SplitStream: High-bandwidth multicast in cooperative environments," in *Proceedings of SOSP'03*, pp. 298–313, 2003.

[9] Cha, M., Kwak, H., Rodriguez, P., Ahn, Y.-Y., and Moon, S., "I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System," in *IMC'07*, pp. 1–14, 2007.

[10] Chen, Y., Katz, R. H., and Kubiatowicz, J., "Dynamic Replica Placement for Scalable Content Delivery," in *Proceedings of IPTPS'02*, pp. 306–318, 2002.

[11] Dabek, F., Kaashoek, M., Karger, D., Morris, R., and Stoica, I., "Wide-area Cooperative Storage with CFS," in *Proceedings of ACM SOSP'01*, pp. 202–215, 2001.

[12] Douceur, J. R., "The Sybil Attack," in *Proceedings of IPTPS'02*, pp. 251–260, 2002.

[13] Douceur, J. R. and Wattenhofer, R. P., "Large-Scale Simulation of Replica Placement Algorithms for a Serverless Distributed File System," in *Proceedings of MASCOTS'01*, pp. 311–319, 2001.

[14] Dumitriu, D., Knightly, E., Kuzmanovic, A., Stoica, I., and Zwaenepoel, W., "Denial-of-service resilience in peer-to-peer file sharing systems," *SIGMETRICS Perform. Eval. Rev.*, vol. 33, no. 1, pp. 38–49, 2005.

[15] Freedman, M. J., Freudenthal, E., and Mazières, D., "Democratizing Content Publication with Coral," in *NSDI'04*, pp. 18–18, 2004.

[16] Ghodsi, A., Alima, L. O., and Haridi, S., "Symmetric Replication for Structured Peer-to-Peer Systems," in *Proceedings of DBISP2P'05*, pp. 74–85, 2005.

[17] Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S., and Stoica, I., "The Impact of DHT Routing Geometry on Resilience and Proximity," in *Proceedings of SIGCOMM'03*, pp. 381–394, 2003.

[18] Gupta, A., Liskov, B., and Rodrigues, R., "Efficient Routing for Peer-to-Peer Overlays," in *Proceedings of NSDI'04*, 2004.

[19] Gupta, I., Birman, K., Linga, P., Demers, A., and Renesse, R. V., "Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead," in *Proceedings of IPTPS '03*, pp. 160–169, 2003.

[20] Hansell, S., "Comcast's Concession to Net Neutrality." http://bits.blogs.nytimes.com/2008/04/17/comcasts-concession-to-net-neutrality/?hp. Last accessed: April 2008.

[21] Harvesf, C. and Blough, D. M., "The Effect of Replica Placement on Routing Robustness in Distributed Hash Tables," in *Proceedings of P2P'06*, pp. 57–66, 2006.

[22] Karagiannis, T., Rodriguez, P., and Papagiannaki, K., "Should Internet Service Providers Fear Peer-Assisted Content Distribution?," in *Proceedings of IMC'05*, pp. 1–14, 2005.

[23] Keromytis, A., Misra, V., and Rubenstein, D., "SOS: Secure Overlay Services," in *Proceedings of SIGCOMM'02*, pp. 61–72, 2002.

[24] Lian, Q., Chen, W., and Zhang, Z., "On the Impact of Replica Placement to the Reliability of Distributed Block Storage Systems," in *Proceedings of ICDCS'05*, pp. 187–196, 2005.

[25] Manku, G. S., Bawa, M., and Raghavan, P., "Symphony: Distributed Hashing in a Small World," in *Proceedings of USITS03*, 2003.

[26] Maymounkov, P. and Mazières, D., "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," in *Proceedings of IPTPS'02*, pp. 53–65, 2002.

[27] Mickens, J. W. and Noble, B. D., "Concilium: Collaborative Diagnosis of Broken Overlay Routes," in *Proceedings of DSN'07*, pp. 225–234, 2007.

[28] On, G., Schmitt, J., and Steinmetz, R., "The Effectiveness of Realistic Replication Strategies on Quality of Availability for Peer-to-Peer Systems," in *Proceedings of P2P'03*, pp. 57–64, 2003.

[29] Plaxton, C. G., Rajaraman, R., and Richa, A., "Accessing Nearby Copies of Replicated Objects in a Distributed Environment," in *Proceedings of ACM SPAA'97*, pp. 311–320, 1997.

[30] Portmann, M., Ardon, S., and Seneviratne, A., "Mitigating Routing Misbehaviour of Rational Nodes in Chord," in *Proceedings of SAINT'04*, pp. 541–545, 2004.

[31] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Schenker, S., "A Scalable Content-Addressable Network," in *Proceedings of SIGCOMM'01*, pp. 161–172, 2001.

[32] Rowstron, A. and Druschel, P., "Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems," in *Proceedings of ACM Middleware'01*, pp. 329–350, 2001.

[33] Saxena, N., Tsudik, G., and Yi, J. H., "Admission Control in Peer-to-Peer: Design and Performance Evaluation," in *Proceedings of SASN'03*, pp. 104–113, 2003.

[34] Seetharaman, S. and Ammar, M., "Characterizing and Mitigating Inter-domain Policy Violations in Overlay Routes," in *Proceedings of ICNP'06*, pp. 259–268, 2006.

[35] Singh, A., Castro, M., Druschel, P., and Rowstron, A., "Defending Against Eclipse Attacks on Overlay Networks," in *Proceedings of ACM SIGOPS'04*, pp. 115–120, 2004.

[36] Sit, E. and Morris, R., "Security Considerations for Peer-to-Peer Distributed Hash Tables," in *Proceedings of IPTPS'02*, pp. 261–269, 2002.

[37] Srivatsa, M. and Liu, L., "Vulnerabilities and Security Threats in Structured Peer-to-Peer Systems: A Quantative Analysis," in *Proceedings of IEEE ACSAC'04*, pp. 252–261, 2004.

[38] Stavrou, A., Keromytis, A., and Rubenstein, D., "Exploiting Structure in DHT Overlays for DoS Protection," tech. rep., Columbia University, 2004.

[39] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H., "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," in *Proceedings of ACM SIGCOMM'01*, pp. 149–160, 2001.

[40] Wallach, D. S., "A Survey of Peer-to-Peer Security Issues.," in *Proceedings of International Software Security Symposium*, pp. 42–57, 2002.

[41] White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., and Joglekar, A., "An Integrated Experimental Environment for Distributed Systems and Networks," in *Proceedings of OSDI'02*, pp. 255–270, 2002.

[42] Xie, H., Krishnamurthy, A., Silberschatz, A., and Yang, Y. R., "P4P: Explicit Communications for Cooperative Control Between P2P and Network Providers." P4P Working Group Whitepaper, 2007.

[43] Zhao, B. Y., Huang, L., Stribling, J., Rhea, S. C., Joseph, A. D., and Kubiatowicz, J. D., "Tapestry: A Resilient Global-scale Overlay for Service Deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 41–53, 2004.

[44] Zhou, S., Ganger, G., and Steenkiste, P., "Location-based Node IDs: Enabling Explicit Locality in DHTs," tech. rep., Carnegie Mellon University, 2003.