EFFICIENT LOCATION-AWARE NODE AND OBJECT DISCOVERY IN LARGE-SCALE NETWORKS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Bernard Wong

August 2011

EFFICIENT LOCATION-AWARE NODE AND OBJECT DISCOVERY IN LARGE-SCALE NETWORKS

Bernard Wong, Ph.D.

Cornell University 2011

The performance of many distributed systems is highly sensitive to the latency of finding objects in response to user requests. Efficient discovery of nodes and objects in the network that satisfy application-specific requirements is therefore a critical building block for many distributed systems.

In this thesis, I introduce a space-based approach to solving node and object discovery problems. This approach represents the relationship between nodes and objects as distances in an abstract space, maps optimization objectives and constraints of the problem to regions in the space, and combines these regions to identify the solution to the discovery problem. Using the space-based approach, I address three common problems involving node and object discovery.

First, I tackle the problem of efficiently discovering nodes with specific network latency characteristics, such as finding the closest server node to a target. This problem is commonly encountered in content distribution networks, online games, and other network services that demand low latency. I describe a system, called Meridian, that uses overlay routing in a small-world inspired network to solve such problems efficiently and accurately.

Second, I address the decentralized approximate search problem, where the objective is to efficiently scan an online database for the set of objects that are most similar to given search terms. I describe the Cubit system, which provides a fully decentralized and efficient approximate search primitive for peer-to-peer

systems.

Finally, I solve the problem of accurately determining the physical location of Internet hosts and describe the Octant system, which uses a novel geometric technique to determine a target node's location from constraints extracted from network measurements.

I characterize the performance and accuracy of these systems with data and evaluations drawn from deployments on PlanetLab and end-user systems. The results show that these space-based systems are accurate, efficient and scalable.

BIOGRAPHICAL SKETCH

Bernard was born in Hong Kong and raised in Toronto, Canada. He received his Bachelor of Applied Science in Computer Engineering from the University of Waterloo in 2003. His interest in distributed systems led to graduate work with his advisor Emin Gün Sirer at Cornell University, where he received his Master of Science in Computer Science in 2007 and his Doctor of Philosophy in Computer Science in 2011.

This thesis is dedicated to my parents. Their love and support was what made this possible.

ACKNOWLEDGEMENTS

I want to thank the many people that played pivotal roles during my doctoral studies and helped shape me into the researcher and person that I am today.

I especially want to thank my Ph.D advisor, Emin Gün Sirer. My words here can only understate my gratitude for the guidance and advice he has given me over my tenure at Cornell. His enthusiasm, energy, and stubborn unwillingness to accept second best were instrumental in motivating me to pursue every idea, build and deploy lots of systems, and become a well-rounded researcher.

I would also like to thank my thesis committee members, Eva Tardos and David Delchamps, and the many researchers and professors I have closely interacted or collaborated with, namely, Aleksandrs Slivkins, Bruce Maggs, David Lie, Paul Francis, Ravi Jain, Robbert van Renesse, Fred Schneider, and Hakim Weatherspoon. I have enjoyed every discussion I have had with them and their encouragement, advice, and insightful comments have greatly broadened my research perspective. I am very grateful to have had them as additional mentors during my doctoral studies.

I also had the good fortune to collaborate closely with a number of amazing students. I would like to thank Nicole Caruso, Phillipa Gill, Ivan Stoyanov, Tushar Arora, Renato Fischer, Robert Escriva, and Ji Yong Shin. I benefited greatly from these collaborations, and I hope I was able to leave a similar impact on their graduate studies.

My years at Cornell would not have been the same without my colleagues in the Systems Lab. I would like to thank Deniz Altinbuken, Willem de Bruijn, Mahesh Balakrishnan, Hitesh Ballani, Tuan Cao, Nicole Caruso, Robert Escriva, Saikat Guha, Qi Huang, Oliver Kennedy, Jed Liu, Ryan Peterson, Patrick

Reynolds, Alan Shieh, Robert Surton, Vidhyashankar Venkataraman, Vivek Vishnumurthy, and Dan Williams for making the lab bearable in the depths of Ithaca's dark and dreary winters.

Finally, I would like to thank my parents. Their unwavering love and support throughout my life was what made this and everything else in my life possible.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Node and object discovery, that is, locating nodes and objects that meet application-specific requirements, is a critical building block for many distributed systems. For instance, the performance of large-scale content distribution networks relies on directing traffic through servers that have low latencies to clients. Similarly, online services must be able to quickly locate user-specific data in the cloud in order to service time-critical user requests, with data retrieval performance often determining the quality of the user experience. Lastly, discovering the geographic location of clients enables advertisers to serve geographically relevant content to their clients. These examples illustrate the ubiquity of node and object discovery problems in today's applications and services.

In this thesis, I will introduce three systems that address problems in node and object discovery common to many real-world applications and services. The first is Meridian, a system for performing efficient latency-aware node discovery, with applications in high performance content delivery as well as many other domains where latency-based node selection plays a role. The second is Cubit, a system for performing accurate, efficient and scalable distributed keyword search that can discover objects with keywords that are close to the search keyword; a capability that addresses the search demands of decentralized content distribution networks. Finally, I will introduce Octant, a system for accurately discovering the geographical location of Internet hosts.

These systems share a common *space-based* approach: they represent the relationship between the nodes and objects as distance in an abstract space, map the optimization objectives and constraints of the problem as regions in the space,

and combine these regions to identify the *solution regions* that correspond to solutions to the problem. Given the solution regions, the systems perform additional problem-specific operations to fully address their respective problems. For example, in latency-aware node discovery, the system must also efficiently discover the nodes that are closest to or within the solution regions. In the next sections, I describe, in turn, the three problems addressed in the thesis, provide the context for the problems and outline the solution strategies, centered on a space-based approach, for solving each of these problems. I describe the concrete systems built based on these strategies.

1.1 Network-Aware Node Discovery

The performance and feasibility of a large-scale service often hinge on its ability to effectively make use of the location of the nodes in the network in its construction and core operations. Services such as content distribution networks and multi-player online games can reduce user-perceived latencies by serving clients from nearby servers and grouping nearby players to the same game instance, respectively.

A precise and compact description of a node's location that captures important network dynamics, such as network latencies to other nodes in the network, in a large-scale network is therefore critically important to these services. However, such a description is surprisingly difficult to implement in practice. A description based on the node's physical location, such as its position on the globe, is compact but fails to capture any network specific details, such as congestion or topology, providing only a coarse-gain estimate of latencies between nodes. A detailed description of a node that includes its network connectivity

information is precise, but it is also cumbersome to manage, update and use, and it still fails to capture the effects of network dynamics, such as congestion and routing delays, on network latencies.

To address the limitations of these simple node location descriptions, there have been significant efforts [69, 25] towards embedding network latencies into simple virtual spaces, such as Euclidean spaces and spherical surfaces, that describe the network locations of nodes as positions in the virtual spaces. These systems, known collectively as *Virtual Coordinates*, assign coordinates, which represent positions, to nodes based on a small number of latency measurements among these nodes. A distance function can then compute the estimated latency between any two nodes based on their coordinates, but at the cost of significant embedding errors. Some of these coordinate systems also exhibit unusual long-term behaviors, such as constant rotation and expansion of the virtual space, leading to instabilities in real deployments [54].

In this thesis, I present Meridian, a new framework for performing scalable, efficient, and accurate node discovery based on network locations. Unlike Virtual Coordinate systems that must first embed latencies into simple virtual spaces, Meridian uses the latencies directly to perform node discovery. A node's location in Meridian is instead defined relative to its latency to other nodes in the system. This intuitive location definition is equivalent to defining a node's position in respect to the *latency space*, an abstract space encapsulating all nodes in the network with network latency used as the distance between node pairs. The latency space enables disparate node discovery problems to be represented in a uniform way.

Following the space-based approach to solving node and object discovery

problems, Meridian maps node discovery constraints and optimization objectives to regions in the latency space, which together, define a solution region. Given this representation of the problem, the goal of node discovery is to find nodes that are within or closest to the solution region in the latency space. A centralized server with access to the inter-node latencies for all nodes in the network can quickly and easily accomplish this goal. However, centralized approaches entail significant overhead in large-scale networks, requiring the central entity to measure and track $O(N^2)$ latencies.

The Meridian framework instead takes a distributed approach to node discovery. In lieu of referencing a centralized repository to find nodes within or near the solution region, Meridian creates a loosely-structured overlay network that facilitates the discovery of a multi-hop path towards the solution region without requiring global information or coordination. Each Meridian node selects $O(\log N)$ other nodes in the network as its overlay peers, with the distances to the peers conforming to a small-world [49] inspired distribution. This peer selection criteria ensures that each Meridian node is an expert of its own region of space, with exponentially fewer peers that are further away. Navigating the overlay involves nodes handing off discovery requests to the peers that are closest to the solution region. Each query hand-off delivers the discovery request to a node with peers that, due to the overlay structure, are closer to solution region than the previous node's peers, facilitating the discovery of additional hand-offs. Both theoretical and empirical results show that the Meridian navigation protocol converges to a node that is closest to or within the solution region in $O(\log D)$ hops, where D is the diameter of the latency space, with high probability.

To completely eliminate the dependence on centralized latency repositories or embedding-based latency prediction, Meridian performs direct latency measurements to satisfy the main requirement of its navigation protocol, namely, determining which peer to forward a discovery request off to. Using direct measurements allows Meridian to avoid embedding errors without large centralization overhead. A naive approach to determining the closest peer to the solution region is to request latency measurements from every peer. The Meridian framework can significantly reduce the number of required latency measurements by pruning peers that are unlikely to be the closest to the solution region. This optimization reduces Meridian's measurement overhead without affecting node discovery accuracy.

I evaluate Meridian in both a PlanetLab deployment, and in simulations. Compared to previous approaches based on virtual coordinates, Meridian can solve node discovery problems with an order-of-magnitude less error. In the median, Meridian is able to find the closest node in a global DNS server dataset with less than one millisecond error. This is in contrast to Virtual Coordinate systems that exhibit errors on the order of tens of milliseconds. Meridian is also scalable, requiring network traffic that grows logarithmically with the system size in order to maintain the loosely structured overlay. Finally, Meridian is resilient to system dynamics, with long-term deployments that do not exhibit the stability problems found in virtual coordinate systems.

1.2 Approximate Keyword Matching

Efficient keyword search is an essential primitive for many large-scale distributed systems, and exact keyword lookup, a specific instance of the keyword search problem, has been the subject of much previous work [78, 91, 106, 76, 62, 45]. However, exact keyword lookups are only marginally effective in applications that accept user contributed content and search queries, such as peer-to-peer content distribution networks, as keywords are not vetted for correctness or uniformity. Common variations and misspellings of the keywords will lead to partial or empty search results. Instead, *approximate keyword matching*, that is, discovering the set of objects whose keywords are close but not identical to the search keyword, is needed to meet the search demands found in unsupervised environments.

Naive approaches to layer approximate keyword matching on top of an exact distributed keyword lookup system, by inserting each object under all possible key variations or issuing queries in parallel with all variants of the search key, lead to highly inefficient solutions. A variation of this approach is to preprocess keywords with Soundex [105], an algorithm for indexing names by sound, groups similar sounding keywords together into a more general keyword for insertion and retrieval. This grouping enables limited retrieval of non-exact keyword matches, but is only effective for phonetic variations of keywords.

An alternative approach to performing approximate keyword matching is to forgo the efficiency of exact distributed lookup systems and instead build on top of unstructured peer-to-peer systems, such as Gnutella, which provide a general search primitive based on query broadcast. Nodes in these systems

scan through their entire local data repository for data that match each search request. Replacing the matching mechanism with one based on a fuzzy similarity metric would therefore yield approximate matches. Such broadcast-based approaches are accurate but not scalable to large deployments, as they may take up to N hops in the worst case, where N is the number of hosts, and place a superlinear aggregate load on the network.

In this thesis, I present Cubit, a scalable peer-to-peer system based on the space-based framework that forms the core of this thesis. The central insight in Cubit is the creation of a keyword space, using edit-distance as the distance function, that captures the relative similarity of keywords and encapsulates both the keywords and node identifiers in the system. Cubit assigns objects to their closest nodes, where distance is a measure of the edit-distance between the object's keywords and the node's identifier. This effectively partitions the keyword space, with each partition centered on a node and representing the region that is closer to this node than any other node in the system, known as the node's *authoritative region*; each node is responsible for storing and retrieving all objects residing in its authoritative region. The keyword space and the partitioning of the space into authoritative regions enable common approximate search queries, such as finding all objects with keywords within one edit-distance of the search keyword, to be represented as a *search region*, and require only discovering and contacting nodes with authoritative regions that overlap or reside within the search region.

Much like Meridian, Cubit nodes keep track of $O(\log N)$ peers in a small-world inspired distribution in the keyword space. This ensures that Cubit nodes have authoritative knowledge about their local regions and have sufficient out-

pointers to retrieve information from more authoritative nodes in other regions. For each search query, a Cubit node first examines its peers, selecting those that are in the solution region or are closer to the solution region than itself. It then retrieves from each selected peer additional candidate nodes that meet the same criteria relative to the peer. The search is repeated with these new candidates that have more information in the proximity of the solution region than the previous selected peers.

As user-submitted search queries typically consist of multiple keywords, Cubit supports a multi-word similarity metric for matching search queries to objects. Cubit also supports combining keywords using user-specified boolean operators, enabling users to precisely define their search queries. This is done efficiently by leveraging probabilistic data-structures to represent, disseminate, and operate on relatively large intermediate result sets with low, constant size overheads.

Providing practical object storage and discovery requires that Cubit handle challenges in object availability and non-uniform search patterns. Objects in Cubit are replicated to the k closest nodes to the object, where k is a system defined parameter, to ensure that objects persist in the presence of node failures. Cubit also introduces a novel load-balancing mechanism to offload query request load for nodes that are responsible for popular objects.

I evaluate Cubit through both a real deployment in a search plugin for Azureus, a popular BitTorrent client, and large-scale simulations. Cubit outperforms approaches that layer approximate matching on top of exact lookup systems, requiring an order-of-magnitude fewer network operations than the naive approach of searching for all possible variations in parallel, and can suc-

cessfully answer 30% more queries than systems that preprocess keywords and search terms with Soundex. In particular, Cubit can place the target object in the top 20 search results for more than 94% of the queries even with a high degree of perturbation to the terms in the search queries.

1.3 Geolocalization of Internet Hosts

Discovering the geographic location of nodes, commonly known as geolocalization, is crucial to the success of many online services. Knowing the geographic location of nodes enables serving customized content to clients, simplifies network management, and improves the fidelity of network diagnosis. Accurately determining the geographic location of a node, however, is difficult as IP addresses do not embed any physical location information; their method of assignment only facilitates efficient network routing.

Past approaches to geolocalization are primarily based on either mining registration databases for IP address to location associations, or multilateration, a technique similar to triangulation, using network latency measurements.

Registration databases, such as the WHOIS database or user profile databases from online services, contain data that associate IP addresses with user-submitted physical locations. However, the accuracy of the user-submitted data is suspect; users may not want to disclose their real locations or may want to associate their IP addresses with the location of their businesses rather than the location of their servers. Furthermore, the number of IP addresses in these databases covers only a small fraction of the public IP address space. Additional ad-hoc techniques must be used to extend the IP address space coverage by clus-

tering IP addresses that may be in nearby locations, which introduce additional geolocalization inaccuracies.

Multilateration, commonly used in radar-based navigation systems, computes the location of a target based on its latency to three or more known landmarks. Measurements from each landmark create *feasibility regions*, in the shape of disks centered at each landmark, that bound where the targets can be located; the distance a measurement probe travels determines the radius of the feasibility region. Multilateration can locate targets to exact positions if the measurement probes travel at a uniform speed via direct great-circle paths between the landmarks and targets. However, this criteria is not satisfied in wired networks due to circuitous packet routing and arbitrary queuing delays at intermediate routers. These non-ideal network properties increase measured latencies and create feasibility regions that are far larger than necessary to enclose the targets. Practical systems based on multilateration can therefore only locate targets to large geographic regions by taking the intersection of very conservative feasibility regions.

In this thesis, I present Octant, a novel framework for accurate and precise geolocalization. Octant provides an intuitive and generic framework that represents constraints, which specify where nodes may or may not be located, as precise geometric regions. Nearly all previous geolocalization approaches can be expressed in terms of these geometric regions and can be incorporated into the framework. The introduction of constraints that exclude geometric regions, known as negative constraints, enables Octant to extract additional information from network measurements. Octant maps nodes to small regions on the globe by solving a system of of geometric constraints. These regions encompass the

nodes' actual locations with high probability.

Unlike the other node and object discovery problems described earlier in this thesis, where the constraints are specified in terms of the distance metric of an abstract space (such as latency and edit-distance), most geolocalization constraints are specified in terms of latencies that must be mapped to geographical distances. Performing this mapping accurately is challenging due to uncertainties introduced by network congestion, indirect routes, and inelastic delays at the last hop.

The Octant framework introduces several novel techniques to limit the effects of these real-world obstacles to geolocalization. These techniques enable the extraction of constraints that are an order of magnitude tighter and more precise than past work. For example, Octant makes use of past measurements between landmarks to generate latency to distance mappings tailored to individual landmarks and latency ranges. These mappings are used in place of idealized transmission models and take into account persistent inflation in latencies seen at specific landmarks. Octant also minimizes the impact of circuitous routing, where packets take paths that significantly deviate from the direct great-circle paths, by breaking paths into straight-line segments. The routers at the end of each segment are geolocalized in turn from the landmarks, with the router from the previous segment used to localize the next router. This piecewise localization can compensate for the delays introduced by indirect routes. Finally, poor last-mile connectivity from the Internet backbone to end users results in last-hop latency dilations that are independent of the geographical distance. Octant captures these inelastic delays separately as an extra "height" dimension in the latency to distance mapping, ensuring that constraints derived

from latencies are unaffected by these delays.

Octant can also incorporate additional constraints from non-latency sources to further refine the geolocalization. For example, geographic and demographic information can derive constraints that eliminate bodies of water that are unlikely to contain the target. Constraints can also be derived from the DNS names of routers that often embed geographic information. By representing constraints as geometric regions, Octant can seamlessly combine all of the available geographic information geometrically to geolocalize a target.

Octant follows the space-based approach by incorporating the different constraints into a feasibility region containing the geolocalization target. However, with the many avenues to extract geographic constraints, there is a high probability that some of these constraints are erroneous. To address this issue, Octant introduces a weight assignment mechanism to characterize the confidence of different constraints. A constraint's weight amplifies or dampens its contribution by estimating the location region of the node of interest. By incorporating weights in the constraint satisfaction process, Octant not only yields the set of feasible points where the node can potentially lie, but also the associated probability for the node residing at each point.

These techniques and mechanisms for extracting precise constraints from the network, for integrating constraints from non-latency data-sources, and for handling different sources of error and imprecision, allow Octant to approach the limits of geolocalization accuracy in wired networks between uncoordinated and unsynchronized hosts. In our PlanetLab deployment, Octant can geolocalize nodes in the U.S. within 22 miles of their actual locations in the median. This result is more than three times better than the previous state-of-the-art.

1.4 Deployment Summary

My work has looked at several common problems concerning node and object discovery that motivated the development of three large-scale and reliable distributed systems. These systems address real user demands and have produced implementations and live deployments that are useful to other researchers as well as general end-users.

The Meridian framework has been adopted by a number of different online services and serves as the primary alternative to virtual coordinates for solving latency-aware node discovery problems. For example, it is used in CobWeb, a large-scale decentralized content distribution network, to minimize latency by directing users to their closest CobWeb nodes. It is deployed in Burrow [48], a wide-area virtual private network, for assigning clients to their closest network gateway nodes. Meridian serves as the node selection mechanism in large-scale DNS redirection services, such as OASIS and ClosestNode.com, that alter their domain name to IP mappings based Meridian's node discovery results [1]. Beyond providing client to server mapping services, Meridian is used in the construction of locality-aware overlay networks [81, 82] that minimize high-latency overlay links. Meridian showed that, for many applications, the latency-prediction primitive offered by virtual coordinate systems is unnecessary and introduces significant inaccuracies compared to Meridian's approach based on direct measurements.

Cubit was implemented as a plug-in for Azureus, a popular BitTorrent client, where it was amongst the first fully decentralized search services for BitTorrent,

[1]The original implementation of OASIS was based on virtual coordinates. However, due to poor prediction accuracy, OASIS was re-implemented to use Meridian for node discovery.

and is still the only decentralized search service to provide approximate keyword search with complete coverage of the nodes in the network.

Octant has garnered significant interest in both the academic community and in industry. It has been used as the basis for work in determining the vulnerability of state-of-the-art network-measurement based geolocalization systems [36]. The techniques in Octant are also used in Alidade, a project I helped start in collaboration with Akamai Technologies. Alidade has two main goals: to extend the scalability of the Octant framework by refactoring the system into a series of Map/Reduce [28] stages, and to build a framework that uses passive background measurements collected by the Akamai content distribution network, rather than use active measurement probes.

Having impact on end-users and other researchers through the use and adoption of my work was one of the primary goals for implementing and deploying these systems. The feedback and data collected from these deployments have also been invaluable in improving the design of the systems to more closely reflect the needs of end-users and the research community.

1.5 Outline

The following chapters describe the contributions of this thesis in detail. Chapter 2 presents Meridian, a system for discovering nodes based on their network location. Chapter 3 describes the problem of performing decentralized object discovery based on approximate keyword matching and presents Cubit, a system using the space-based approach to address this problem. Chapter 4 presents Octant, a system for performing accurate geolocalization of Internet

hosts. Chapter 5 provides a summary of work related to the topics discussed in this thesis. Finally, Chapter 6 summarizes the contributions of this thesis and outlines future directions.

CHAPTER 2

NETWORK LOCATION-AWARE NODE SELECTION

Selecting nodes based on their location in the network is a basic building block for many high-performance distributed systems. In small systems, it is possible to perform extensive measurements and make decisions based on global information. For instance, in an online game with few servers, a client can simply measure its latency to all servers and bind to the closest one for minimal response time. However, collecting global information is infeasible for a significant set of recently emerging large-scale distributed applications, where global information is unwieldy and lack of centralized servers makes it difficult to find nodes that fit selection criteria. Yet many distributed applications, such as filesharing networks, content distribution networks, backup systems, anonymous communication networks, pub-sub systems, discovery services, and multi-player online games could benefit substantially from selecting nodes based on their location in the network.

This chapter introduces a lightweight, scalable and accurate framework, called Meridian, for performing node selection based on network location [1]. Meridian forms a loosely-structured overlay network, uses direct latency measurements instead of latency estimates from virtual coordinates, and can solve spatial queries without an absolute coordinate space. Meridian provides the general framework for solving frequently encountered location-related problems in distributed systems. Specifically, this thesis examines how Meridian can be used to efficiently find the closest node to a target, the latency minimizing

[1] We use the term "location" to refer to a node's placement in the Internet as defined by its round-trip latency to other nodes. While Meridian does not assume that there is a well-defined location for any node, our illustrations depict a single point in a two-dimensional space for clarity.

node to a given set of nodes, and the set of nodes that lie in a region defined by latency constraints. Although it is strictly less general than virtual coordinates, we show that Meridian incurs significantly less error. Empirical results from both simulations parameterized with measurements from a large-scale network study and a PlanetLab deployment show that Meridian is an order of magnitude more accurate than virtual coordinates based systems in performing location-aware node selection. Theoretical analyses of Meridian [101, 87] corroborate these results and show that Meridian provides robust performance, high scalability and good load balance when deployed in metric spaces that closely model Internet latencies.

2.1 Framework

The basic Meridian framework is based around three mechanisms: a loose routing system based on multi-resolution rings on each node, an adaptive ring membership replacement scheme that maximizes the usefulness of the nodes populating each ring, and a gossip protocol for node discovery and dissemination.

2.1.1 Multi-Resolution Rings

Each Meridian node keeps track of a small, fixed number of other nodes in the system, and organizes this list of peers into concentric, non-overlapping rings. The ith ring has inner radius $r_i = \alpha s^{i-1}$ and outer radius $R_i = \alpha s^i$, for $i > 0$, where α is a constant, s is the multiplicative increase factor, and $r_0 = 0$, $R_0 = \alpha$ for the innermost ring. Each node keeps track of a finite number of rings; all rings $i > i^*$ for a system-wide constant i^* are collapsed into a single, outermost ring that

spans the range $[\alpha s^{i^*}, \infty]$.

Meridian nodes measure the distance d_j to a peer j, and place that peer in the corresponding ring i such that $r_i < d_j \leq R_i$. This sorting of neighbors into concentric rings is performed independently at each node and requires no fixed landmarks or distributed coordination. Each node keeps track of at most k nodes in each ring and drops peers from overpopulated rings. Consequently, Meridian's space requirement per node is proportional to k.

The ring structure, with its exponentially increasing ring radii, favors nearby neighbors and enables each node to retain a relatively large number of pointers to nodes in their immediate vicinity. This allows a node to authoritatively answer geographic queries for its region of the network. At the same time, the ring structure ensures that each node retains a sufficient number of pointers to remote regions, and can therefore dispatch queries towards nodes that specialize in those regions. An exponentially increasing radius also makes the total number of rings per node manageably small and i^* clamps it at a constant.

2.1.2 Ring Membership Management

The number of nodes per ring, k, represents an inherent tradeoff between accuracy and overhead. A large k increases a node's information about its peers and helps it make better choices when routing queries. On the other hand, a large k also entails more state, more memory and more bandwidth at each node.

Within a given ring, node choice can have a significant effect on the performance of the system. A set of ring members that are geographically distributed

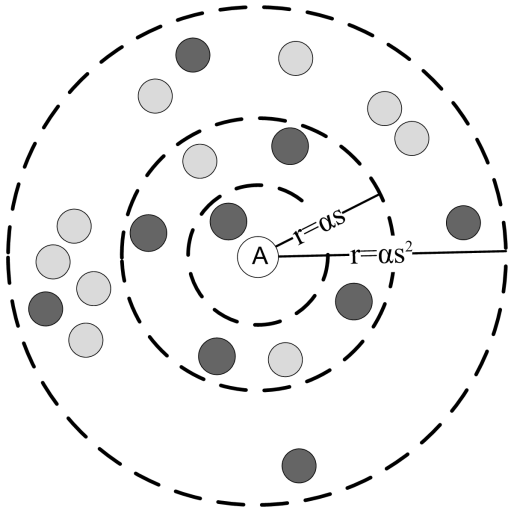


Figure 2.1: Each Meridian node keeps track of a fixed number of other nodes and organizes these nodes into concentric, non-overlapping rings of exponentially increasing radii.

provides much greater utility than a set of ring members that are clustered together, as shown in Figure 2.1. Intuitively, nodes that are geographically diverse instead of clustered together enable a node to reach a larger portion of the network efficiently. Consequently, Meridian strives to promote geographic diversity within each ring.

Meridian achieves geographic diversity by periodically reassessing ring membership decisions and replacing ring members with alternatives that provide greater diversity. Within each ring, a Meridian node not only keeps track of the k primary ring members, but also a constant number l of secondary ring members, which serve as a FIFO pool of candidates for primary ring membership.

We quantify geographic diversity through the hypervolume of the k-polytope formed by the selected nodes. To compute the hypervolume, each

node defines a local, non-exported coordinate space. A node i will periodically measure its distance d^i_j to another node j in the same ring, for all $0 \leq i, j \leq k + l$. The coordinates of node i consist of the tuple $\langle d^i_1, d^i_2, ..., d^i_{k+l} \rangle$, where $d^i_i = 0$. This embedding is trivial to construct and does not require a potentially error-introducing mapping from high-dimensional data to a lower number of dimensions.

Having computed the coordinates for all of its members in a ring, Meridian nodes then determine the subset of k nodes that provide the polytope with the largest hypervolume. For small k, it is possible to determine the maximal hypervolume polytope by considering all possible polytopes from the set of $k + l$ nodes. For large $k + l$, evaluating all subsets is infeasible. Instead, Meridian uses a greedy algorithm: A node starts out with the $k + l$ polytope, and iteratively drops the vertex (and corresponding dimension) whose absence leads to the smallest reduction in hypervolume until k vertices remain. The remaining vertices are designated the new primary members for that ring, while the remaining l nodes become secondaries. This computation can be performed in linear time using standard computational geometry tools [10]. The ring membership management occurs in the background and its latency is not critical to the correct operation of Meridian. Note that the coordinates computed for ring member selection are used only to select a diverse set of ring members; they are not exported by Meridian nodes and play no role in query routing.

Churn in the system can be handled gracefully by the ring membership management system due to the loose structure of the Meridian overlay. If a node is discovered to be unreachable during the replacement process, it is dropped from the ring and removed as a secondary candidate. If a peer node is discov-

ered to be unreachable during gossip or the actual query routing, it is removed from the ring, and replaced with a random secondary candidate node. The quality of the ring set may suffer temporarily, but will be corrected by the next ring replacement. Discovering a peer node failure during query routing can reduce query performance; k can be increased to compensate for this expected rate of failure.

2.1.3 Gossip Based Node Discovery

The use of a gossip protocol to perform node discovery allows the Meridian overlay to be loosely connected, highly robust and inexpensively kept up-to-date of membership changes. This gossip protocol is based on an anti-entropy push protocol [29] that implements a membership service. The central goal of our gossip protocol is for each node to discover and maintain a small set of pointers to a sufficiently diverse set of nodes in the network. Our gossip protocol works as follows:

1. Each node A randomly picks a node B from each of its rings and sends a gossip packet to B containing a randomly chosen node from each of its rings.
2. On receiving the packet, node B determines through direct probes its latency to A and to each of the nodes contained in the gossip packet from A.
3. After sending a gossip packet to a node in each of its rings, node A waits until the start of its next gossip period and then begins again from step 1.

In step 2, node B sends probes to A and to the nodes in the gossip packet from A regardless of whether B has already discovered these nodes. This repinging ensures that stale latency information is updated, as latency between

nodes on the Internet can change dynamically. The newly discovered nodes are placed on B's rings as secondary members.

For a node to initially join the system, it needs to know the IP address of one of the nodes in the Meridian overlay. The newly joining node contacts the Meridian node and acquires its entire list of ring members. It then measures its latency to these nodes and places them on its own rings; these nodes will likely be binned into different rings on the newly joining node. From there, the new node participates in the gossip protocol as usual.

The period between gossip cycles is initially set to a small value in order for new nodes to quickly propagate their arrival to the existing nodes. The new nodes gradually increase their gossip period to the same length as the existing nodes. The choice of a gossip period depends on the expected rate of latency change between nodes and expected churn in the system.

2.1.4 Maintenance Overhead

The average bandwidth overhead to maintain the multi-resolution rings of a Meridian node is modest. The number of gossip packets a node receives is equal to the number of neighbors ($m \log N$) multiplied by the probability of being chosen as a gossip target by one of the neighbors ($\frac{1}{\log N}$), where m is the number of rings in the ring-set. A node should therefore expect to send and receive m gossip packets and to initiate m^2 probes per gossip period. A node is also the recipient of probes from neighbors of its neighbors. Since it has $m \log N$ neighbors, each of which sends m gossip packets, there are $m^2 \log N$ gossip packets with a $\frac{1}{\log N}$ probability of containing a reference to it. Therefore, a node expects

to receive m^2 probes from neighbors of its neighbors. Assuming $m = 9$, a probe packet size of 50 bytes, two packets per probe, and a gossip packet size of 100 bytes, membership dissemination consumes an average of 20.7 KB/period of bandwidth per node. For a gossip period of 60 seconds, the average overhead associated with gossip is 345 B/s, independent of system size.

There is also maintenance overhead for performing ring management. In every ring management period where the membership of one ring is re-evaluated, $2 \log N$ requests are sent, $2 \log N$ are received, $4 \log^2 N$ probes are sent, and $4 \log^2 N$ are received. Assuming two packets are necessary per request and per probe, the size of a probe request packet is 100 bytes and a probe packet is 50 bytes, and a 2000 node system with 16 nodes per ring, ring management consumes an average of 218 KB/period. For a ring management period of 5 minutes, the average overhead associated with ring management is 727 B/s. This analysis conservatively assumes that all primary and secondary rings of all nodes are full, which is unlikely in practice.

2.2 Applications

The following three sections describe how Meridian can be used to solve some frequently encountered location-related problems in distributed systems.

2.2.1 Closest Node Discovery

Meridian locates the closest node by performing a multi-hop search where each hop exponentially reduces the distance to the target. This is similar to searching

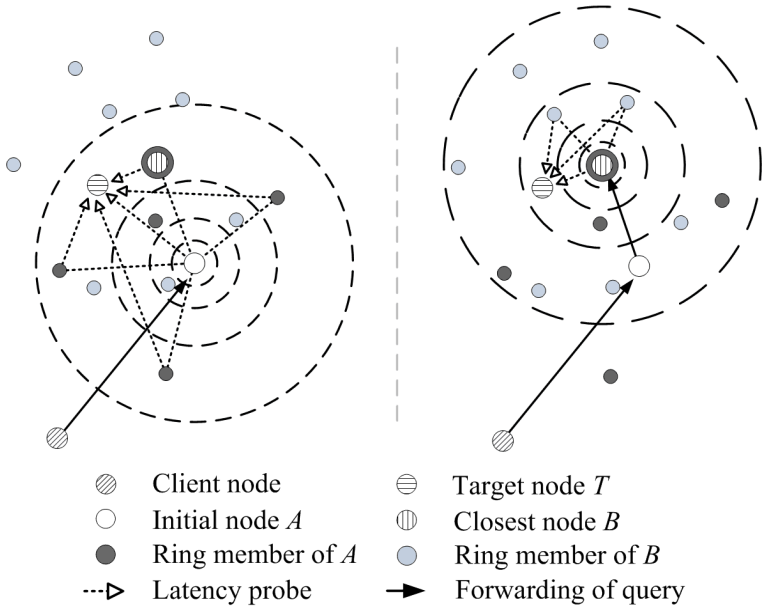


Figure 2.2: A client sends a "closest node discovery to target T" request to a Meridian node A, which determines its latency d to T and probes its ring members between $(1-\beta)\cdot d$ and $(1+\beta)\cdot d$ to determine their distances to the target. The request is forwarded to the closest node thus discovered, and the process continues until no closer node is detected.

in structured peer-to-peer networks such as Chord [91], Pastry [78] and Tapestry [106], where each hop brings the query exponentially closer to the destination, though in Meridian the routing is performed using physical latencies instead of numerical distances in a virtual identifier space. Another important distinction that Meridian holds over the structured peer-to-peer networks is the target node need not be part of the Meridian overlay. The only requirement is that the latencies between the nodes in the overlay and the target node are measurable. This enables applications such as finding the closest node to a public web server, where the web server is not directly controlled by the distributed application and only responds to HTTP queries.

When a Meridian node receives a request to find the closest node to a target, it determines the latency d between itself and the target. Once this latency is determined, the Meridian node simultaneously queries all of its ring members whose distances are within $(1 - \beta) \cdot d$ to $(1 + \beta) \cdot d$. These nodes measure their distance to the target and report the result back to the Meridian node. Nodes that take more than $(2\beta + 1) \cdot d$ to provide an answer are ignored, as they are more than βd away from the target.

Meridian uses an acceptance threshold β to determine the reduction in distance at each hop. The route acceptance threshold is met if one or more of the queried peers is closer than β times the distance to the target, and the client request is forwarded to the closest node. If no peers meet the acceptance threshold, then routing stops and the closest node currently known is chosen. Figure 2.2 illustrates the process.

Meridian is agnostic to the choice of a route acceptance threshold β, where $0 \leq \beta < 1$. A small β value reduces the total number of hops, as fewer peers can satisfy the requirement, but introduces additional error as the route may be prematurely stopped before converging to the closest node. A large β stems errors from both poor neighbor selection and small violations in triangle inequality at the expense of increased hop count.

2.2.2 Central Leader Election

Another frequently encountered problem in distributed systems is to locate a node that is "centrally situated" with respect to a set of other nodes as illustrated in Figure 2.3. Typically, such a node plays a specialized role in the network that

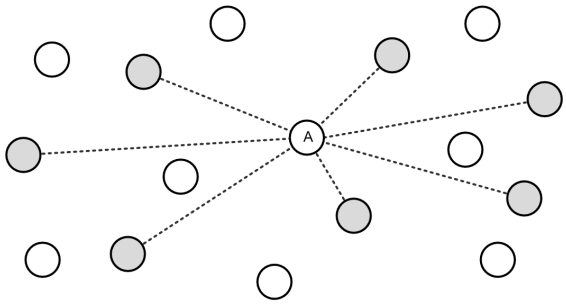


Figure 2.3: Central leader election selects the Meridian node with the minimum average distance to the nodes in the target group. For this example, node A is the the most centrally situated Meridian node (white) to the target nodes (gray).

requires frequent communication with the other members of the set; selecting a centrally located node minimizes both latency and network load. An example application is leader election, which itself is a building block for higher level applications such as clustering and low latency multicast trees.

The central leader election application can be implemented by extending the closest node discovery protocol. We replace d in the single target closest node selection protocol with d_{avg} for central leader election. When a Meridian node receives a client request to find the closest node to the target set T, it determines the latency set $\{d_1, ..., d_{|T|}\}$ between itself and the targets through direct measurements, and computes the average latency $d_{avg} = (\sum_{i=1}^{|T|} d_i)/|T|$. It selects ring members that have latency between $(1-\beta) * \min\{d_1, ..., d_{|T|}\}$ and $(1+\beta) * \max\{d_1, ..., d_{|T|}\}$ to itself, and requests these peers to.determine their respective average latency to the targets. The remaining part of the central leader election application follows exactly from the closest node discovery protocol.

Changing the latency aggregation function from taking the average of the

latencies to the highest latency target is a useful variation to the protocol, as it reduces the difference in latency between the targets to the chosen node. This is useful in multi-player online games, as a player with a significantly lower latency to the game server than the others has an unfair advantage because it is the first to receive and react on game events.

2.2.3 Multi-Constraint System

Another frequent operation in distributed systems is to find a set of nodes satisfying constraints on the network geography. For instance, an ISP or a web hosting service is typically bound by a service level agreement (SLA) to satisfy latency requirements to well-known peering locations when hosting services for clients. A geographically distributed ISP may have thousands of nodes at its disposal, and finding the right set of nodes that satisfy the given constraints may be necessary for fulfilling an SLA. Latency constraints are also important for grid based distributed computation applications, where the latency between nodes working together on a problem is often the main efficiency bottleneck. A customer may want to specify that $\forall q, p \in P$ where P is the set of grid nodes, $d_{q,p} < \gamma$ for some desired latency γ.

Finding a node that satisfies multiple constraints can be viewed as a node selection problem, where the constraints define the boundaries of a region in space (the solution space), as illustrated in Figure 2.4. A constraint is specified as a target and a latency bound around that target. When a Meridian node receives a multi-constraint query with u constraints specified as $\langle target_i, range_i \rangle$, for all $0 < i \leq u$, it measures its latency d_i to the target nodes and calculates its

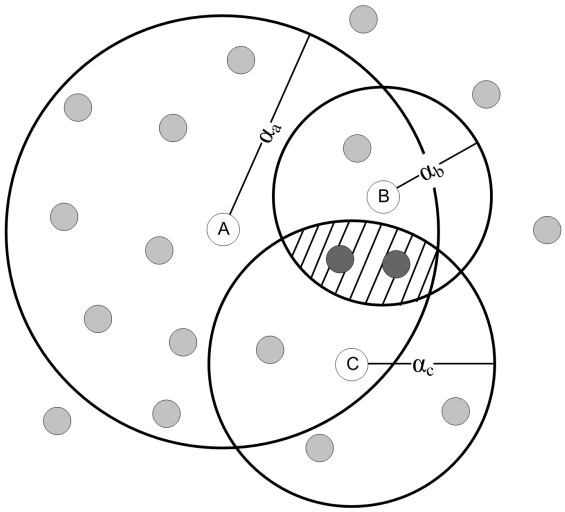


Figure 2.4: A multi-constraint query consisting of targets A, B, C with respective latency constraints of $\alpha_a, \alpha_b, \alpha_C$. The shaded area represents the solution space.

distance to the solution space as

$$s = \sum_{i=1}^{u} \max(0, d_i - range_i)^2 \tag{2.1}$$

If s is 0, then the current node satisfies all the constraints, and it returns itself as the solution to the client. Otherwise, it iterates through all its peers, and simultaneously queries all peers j that are within $\max(0, (1-\beta) \cdot (d_i - range_i))$ to $(1+\beta) \cdot (d_i + range_i)$ from itself, for all $0 < i \leq u$. These nodes include all the peers that lie within the range of at least one of the constraints, and possibly other peers that do not satisfy any of the constraints, but are nevertheless close to the solution space. These peer nodes measure their distance to the u targets and report the results back to the source. Nodes that take longer than $\max_{0<i\leq u}((2\beta + 1) \cdot (d_i + range_i))$ to provide an answer are ignored.

The distance s_j of each node j to the solution space is calculated using the metric s defined above. If s_j is 0, then node j satisfies all the constraints and is returned as a solution to the client. If no zero valued s_j is returned, the client determines whether there is an $s_j < \beta \cdot s$, where β is the route acceptance threshold. If the route acceptance threshold is met, the client request is forwarded to the peer closest to the solution space. A larger β may increase the success rate, at the expense of increased hops.

2.3 Meridian Query Language

We described the Meridian framework and provided three algorithms for solving three commonly encountered problems. But there may well be other location-related problems to solve, and other solution strategies that applications may require. Some applications, such as online games or file backups, may value equal-distance peers for fairness or large distances from anchors to provide uncorrelated failures, in addition to wanting proximity to a target node set. To enable such applications and others that we could not foresee, we added a language for expressing application specific algorithms and a runtime for evaluating such algorithms safely on top of the Meridian framework.

The Meridian Query Language (MQL) is a safe, polymorphic, and dynamically typed variant of *C* that provides tight resource and processing constraints on each query. Every Meridian packet carries the full query specified by the user, similar to a capsule in an active network [94], and the query is executed on each Meridian node that a query traverses.

MQL's grammar and lexical syntax is very similar to *C*. To ensure safety,

there are no pointers nor any direct references to memory, and type checking as well as bounds checking are performed at runtime. MQL has the primitive types *int*, *double* and *string*, as well as two primitive structures *Node* and *Measurement*. The structures are used by many of the library functions that provide access to Meridian operations. The *Node* structure is an abstraction of a Meridian node and contains the address of the node, the Meridian port, and the address and port of an optional proxy node [2]. The *Measurement* structure encapsulates the latency information from a Meridian node to a set of targets, and serves as the return value for library functions that issue latency probes.

An MQL query is processed in an isolated runtime environment which consists of an interpreter that can multiplex multiple simultaneous queries, and a rich set of native library functions for issuing latency probes or accessing the ring structure. The runtime environment enforces local resource constraints, such as the amount of time or memory a query can execute for or allocate per node. It also enforces resource restrictions that span multiple nodes, such as the maximum number of hops per query and the query lifetime in the system, using auditing information embedded into the query packet headers.

The MQL library provides queries access to the underlying Meridian subsystems along with convenience functions that are commonly used in localization queries. It consists of local functions shown in Figure 2.5 for accessing local Meridian ring membership information, mathematical functions, and array operations, as well as remote functions shown in Figure 2.6 for resolving names, issuing probes, and transferring the control flow to another node.

The MQL language and library functions were designed specifically to ad-

[2]Section 2.5.2 describes in detail the use of proxy nodes in Meridian.

```
Node get_self()
Node[] ring_lt(double latency_ms)
Node[] ring_le(double latency_ms)
Node[] ring_gt(double latency_ms)
Node[] ring_ge(double latency_ms)
T print(T value)
T println(T value)
double dbl(int x)
int round(double x)
int ceil(double x)
int floor(double x)
double sin(double x)
double cos(double x)
double tan(double x)
double asin(double x)
double acos(double x)
double atan(double x)
double log(double x)
double exp(double x)
double pow(double x, double y)
void push_back(T array[], T value)
void pop_back(T array[])
int array_size(T array[])
T[] array_intersect(T x[], T y[])
T[] array_union(T x[], T y[])
T array_max(T x[])
int array_max_offset(T x[])
T array_min(T x[])
int array_min_offset(T x[])
double array_avg(T x[])
```

Figure 2.5: Local system functions for accessing local Meridian ring membership information, mathematical functions, and array operations. The type *T* in the function definitions is a generic type that can be instantiated as any primitive or abstract data type at runtime.

```
T rpc(Node target, func, ...)
int dns_lookup(string name)
string dns_addr(int addr)
Measurement[] get_distance_dns(
    Node target[],
    int timeout_ms)
Measurement[] get_distance_dns(
    Node source[],
    Node target[],
    int timeout_ms)
Measurement[] get_distance_tcp(
    Node target[],
    int timeout_ms)
Measurement[] get_distance_tcp(
    Node source[],
    Node target[],
    int timeout_ms)
Measurement[] get_distance_icmp(
    Node target[],
    int timeout_ms)
Measurement[] get_distance_icmp(
    Node source[],
    Node target[],
    int timeout_ms)
Measurement[] get_distance_ping(
    Node target[],
    int timeout_ms)
Measurement[] get_distance_ping(
    Node source[],
    Node target[],
    int timeout_ms)
```

Figure 2.6: System functions for issuing remote procedure calls, DNS name resolutions, and latency probes using DNS queries, TCP SYN/ACK packets, ICMP ECHO packets or custom Meridian UDP packets.

```
 1 Measurement closest(double beta, Node target) {
 2   Node t[] = {target};
 3   Measurement self = get_distance_tcp(t, -1);
 4   double self_lat = self.distance[0];
 5   Node ring_m[] = array_intersect(
 6     ring_ge((1.0 - beta) * self_lat),
 7     ring_le((1.0 + beta) * self_lat));
 8   if (array_size(ring_m) == 0) {
 9     return self;
10   }
11   Measurement r_lat[] = get_distance_tcp(ring_m,
12     t, ceil((2.0 * beta + 1.0) * self_lat));
13   int min_index = -1;
14   double min_lat = self_lat;
15   for (int i=0; i < array_size(r_lat); i=i+1) {
16     double cur_lat = r_lat[i].distance[0];
17     if (cur_lat < min_lat) {
18       min_index = i;
19       min_lat = cur_lat;
20     }
21   }
22   if (min_index == -1) {
23     return self;
24   }
25   Measurement min_n = r_lat[min_index];
26   if (min_n.addr != 0
27       && min_lat < (self_lat * beta)) {
28     Measurement ret_n = rpc(
29       ring_m[min_index], closest,
30       beta, t);
31     if (ret_n.addr != 0) {
32       return ret_n;
33     }
34   }
35   return min_n;
36 }
```

Figure 2.7: The closest node discovery protocol in MQL.

dress problems in the network localization domain. This enables MQL implementations of network localization algorithms to be intuitive and compact. Figure 2.7 illustrates the closest node discovery protocol written in MQL. The MQL version specifies the complete closest node discovery protocol, and is signficantly shorter and easier to understand than our previous hand-crafted C++ version.

2.4 ClosestNode.com: A General-Purpose Deployment

The network localization algorithms we have described so far rely the participation of client applications in the localization protocol. Explicit network localization queries must be added to existing client applications to take advantage of Meridian; an operational challenge for widely deployed software. To enable support for existing client applications and lower the barrier of entry

for new applications, we deployed a DNS to Meridian gateway called ClosestNode.com. This gateway allows Meridian-oblivious clients to perform closest node selection to registered services via DNS requests. For example, a registered service of ClosestNode.com, named *dht*, would be given the sub-domain *dht.closestnode.com*. When a client issues a request to resolve *dht.closestnode.com*, the ClosestNode.com DNS server, which is the authoritative name server for the domain, initiates a modified closest node discovery request on the Meridian overlay specific to the service using the client's DNS server as the target. The IP addresses of the four closest nodes are then returned as the result of the domain resolution to the client. The necessary changes to the service itself are minimal. Service providers need to provide the ClosestNode.com DNS server with a small list of nodes that are running their service, and the service needs to either be modified to call a Meridian library function at startup, or start a stand-alone Meridian daemon along-side it.

Providing more than one nearby node in the domain name resolution helps mitigate the effects of node failures on service availability. To support this requirement, we implemented a *k*-closest node discovery protocol in MQL. Although based on the standard closest node discovery protocol, substantial changes were required to expand the search results. A running list of the closest unexplored candidates nodes is used to ensure good coverage of the search space beyond the closest node. Recursive resolution was replaced with iterative resolution to reduce the reverse path length and resolution latency. These changes illustrate the flexibility of MQL which enabled ClosestNode.com to be deployed with a new network localization protocol without requiring changes to the Meridian binaries. The MQL source code for the *k*-closest node discovery protocol can be found in Appendix A.

For services that are sensitive to resolution latency, ClosestNode.com provides an option to resolve specific domains immediately for new client DNS servers. These results are chosen randomly from the overlay and are returned with low TTLs. At the same time, the ClosestNode.com DNS server initiates closest node discoveries for these client DNS servers and caches the results. The cached results are used when the client DNS servers return after the initial results expire.

CobWeb [89], a distributed web cache and proxy deployed on PlanetLab, is an example of a large-scale service using ClosestNode.com for client redirection. We will show in Section 2.5 the improvements in the end-to-end performance of this service from using ClosestNode.com.

2.5 Evaluation

We evaluated Meridian through both a large scale simulation parameterized with real Internet latencies and a physical deployment on PlanetLab. We also evaluated Meridian's impact on application performance of a PlanetLab service in the context of ClosestNode.com.

2.5.1 Simulation

We performed a large scale measurement study of internode latencies between 2500 nodes to parameterize our simulations. We collected pair-wise round-trip time measurements between 2500 DNS servers at unique IP addresses, spanning 6.25 million node pairs. The study was performed on 10 different Plan-

etLab nodes, with the median value of the runs taken for the round-trip time of each pair of nodes. Data collection was performed on May 5-13, 2004; query interarrival times were dilated, and the query order randomized, to avoid queuing delays at the DNS servers. The latency measurements between DNS servers on the Internet were performed using the King measurement technique [40].

In the following experiments, each test consists of 4 runs with 2000 Meridian nodes, 500 target nodes, $k = 16$ nodes per ring, 9 rings per node, $s = 2$, probe packet size of 50 bytes, $\beta = \frac{1}{2}$, and $\alpha = 1$ms, for 25000 queries in each run. The results are presented either as the mean result of the 100000 total queries, or as the mean of the median value of the 4 runs. All references to latency in this section are in terms of round-trip time. Each simulation run begins from a cold start, where each joining node knows only one existing node in the system and discovers other nodes through the gossip protocol.

We compare Meridian to virtual coordinates computed through network embeddings. We computed the coordinates for our 2500 node data set using GNP, Vivaldi and Vivaldi with height [25]. GNP is a global virtual coordinate system based on static landmarks. We configured it for 15 landmarks and 8 dimensions, and used the N-clustered-medians protocol for landmark selection. Vivaldi is a virtual coordinate scheme based on spring simulations and was configured to use 6 dimensions with 32 neighbors. Vivaldi with height is a recent scheme that performs a non-Euclidean embedding which assigns a two dimensional location plus a height value to each node. We randomly select 500 targets from our data set of 2500 nodes.

We first examine the inherent embedding error in absolute coordinate systems and determine the error involved in finding the closest neighbor. The dark

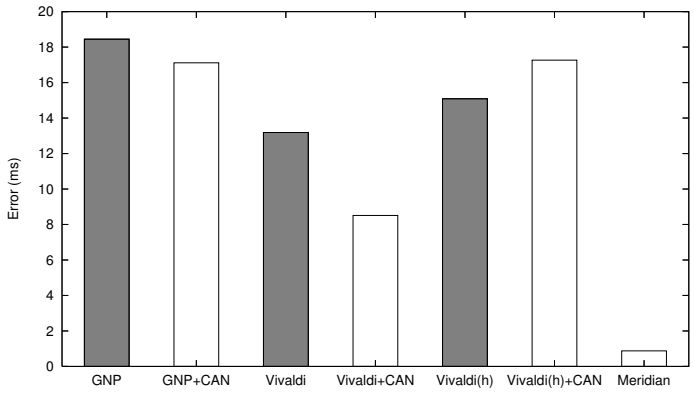


Figure 2.8: Light bars show the median error for discovering the closest node. Darker bars show the inherent embedding error with coordinate systems. Meridian's median closest node discovery error is an order of magnitude lower than schemes based on embeddings.

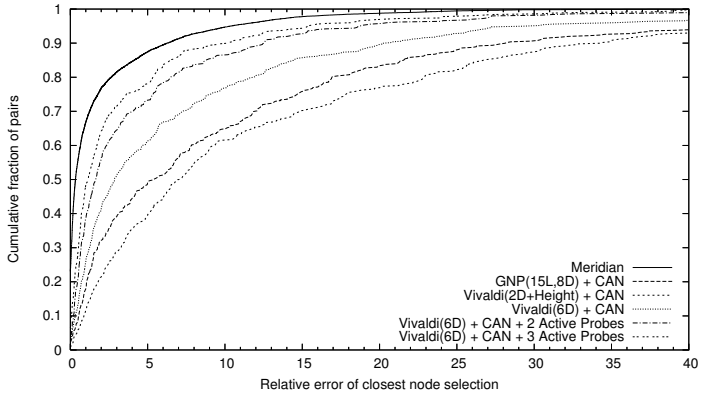


Figure 2.9: Meridian's relative error for closest node discovery is significantly better than virtual coordinates.

bars in Figure 2.8 show the median embedding error of each of the coordinate schemes, where the embedding error is the absolute value of the difference between the measured distance and predicted distance over all node pairs. While these systems incur significant errors during the embedding, they might still

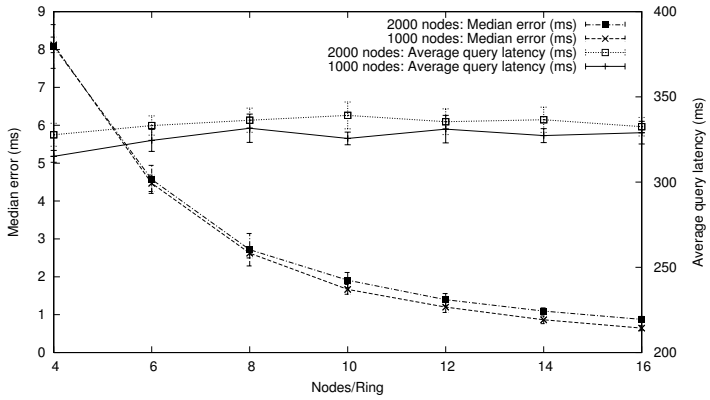


Figure 2.10: A modest number of nodes per ring achieves low error. Average latency is determined by the slowest node in each ring and the hop count, and remains constant within measurement error bounds.

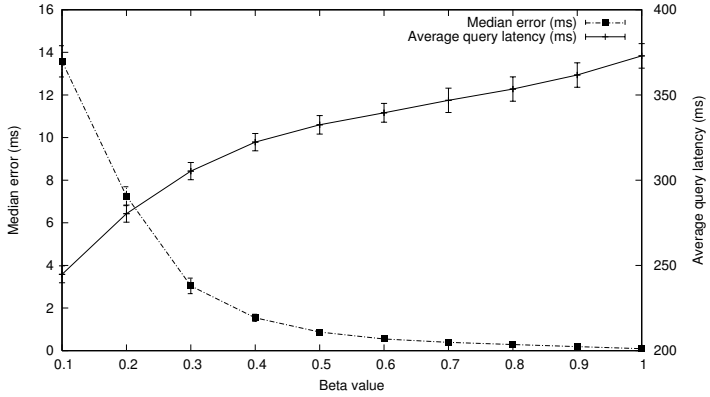


Figure 2.11: An increase in β significantly improves accuracy for $\beta \leq 0.5$. The average query latency increases with increasing β, as a bigger β increases the average number of hops taken in a query.

pick the correct closest node. To evaluate the error in finding the closest node, we assume the presence of a geographic query routing layer, such as a CAN deployment, with perfect information at each node. This assumption biases the experiment towards virtual coordinate systems and isolates the error inherent

in network embeddings. The resulting median errors for all three embedding schemes, as shown by the light bars in Figure 2.8, are an order of magnitude higher than Meridian. Figure 2.9 compares the relative error CDFs of different closest node discovery schemes. Meridian has a lower relative error than the embedding schemes by a large margin over the entire distribution.

We also examine the improvement in closest node discovery accuracy using Vivaldi coordinates with the addition of latency data from active probes. We modify Vivaldi+CAN to return the top M candidates based on their coordinates and actively probe the target to determine the closest candidate. These probes are on top of the those required to determine the virtual coordinates of the target; an on-demand prerequisite for targets that are not already part of the Vivaldi overlay. By default, each new Vivaldi node communicate with 16 nearby neighbors and 16 distant neighbors to calculate its coordinates [25]. Figure 4.8 shows the results for $M = 2$ and $M = 3$. Active probing greatly improves the accuracy of closest node discovery, but is still significantly less accurate than Meridian and requires additional probing on top of those needed to assign coordinates to the target, which already has similar probing requirements as Meridian. Note that selecting the M closest targets for $M > 1$ in a scalable ($< O(N)$) manner requires additional, complex extensions to CAN that are equivalent to a multi-dimensional expanding ring search.

The accuracy of Meridian's closest node discovery protocol depends on several parameters, such as the number of nodes per ring k, acceptance interval β, the constant α, and the gossip rate. The most critical parameter is the number of nodes per ring k, as it determines the coverage of the search space at each node. Figure 2.10 shows that median error drops sharply as k increases. Hence, a node

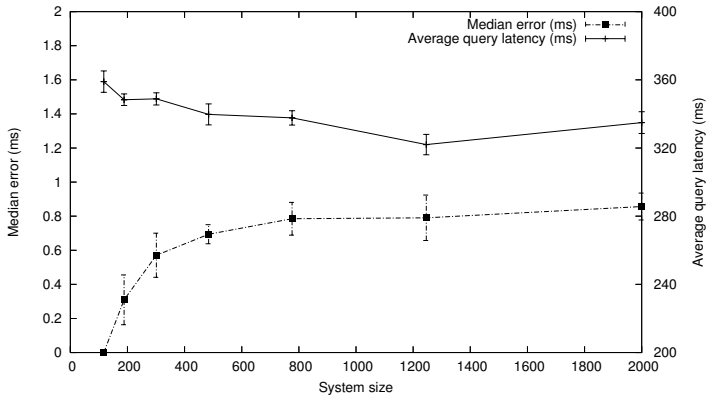


Figure 2.12: Median error and average query latency as a function of system size, for $k = \lfloor \log_{1.6} N \rfloor$; both remain constant as the network grows, as predicted by the analytical results.

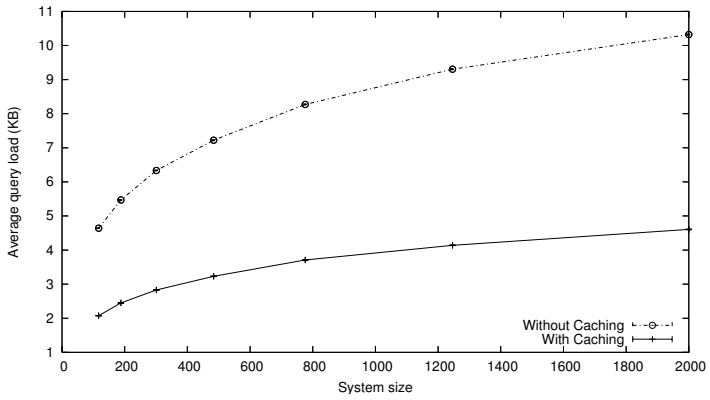


Figure 2.13: The average load of a closest node discovery query increases sub-linearly with system size ($k = \lfloor \log_{1.6} N \rfloor$). With minimal per query caching, average load is reduced by more than half.

only needs to keep track of a small number of other nodes to achieve high accuracy. The results indicate that as few as eight nodes per ring can return very accurate results with a system size of 2000 nodes.

High accuracy must also be coupled with low query latency for interactive applications that have a short lifetime per query and cannot tolerate a long ini-

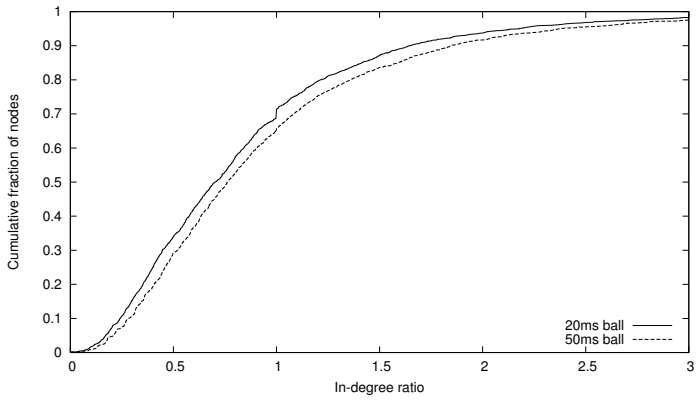


Figure 2.14: The in-degree ratio shows the average imbalance in incoming links within spherical regions. More than 90% of regions have a ratio less than 2.

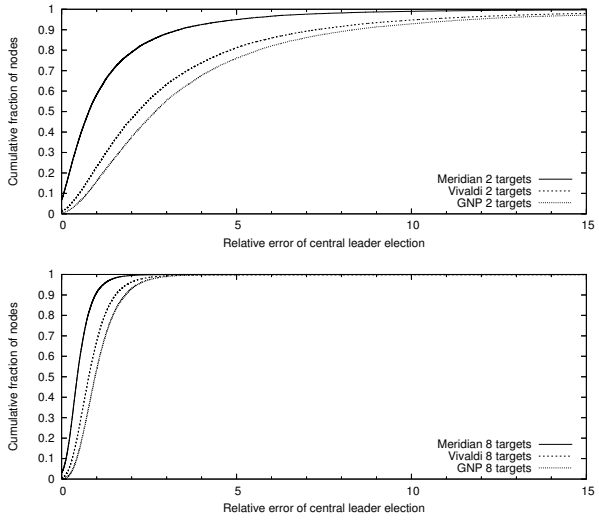


Figure 2.15: Central leader election accuracy.

tial setup time. The closest node discovery latency is dominated by the sum of the maximum latency probe at each hop plus the node to node forwarding latency; we ignore processing overheads because they are negligible in comparison. Meridian bounds the maximum latency probe by $2\beta + 1$ times the latency from the current intermediate node to the destination, as any probe that requires

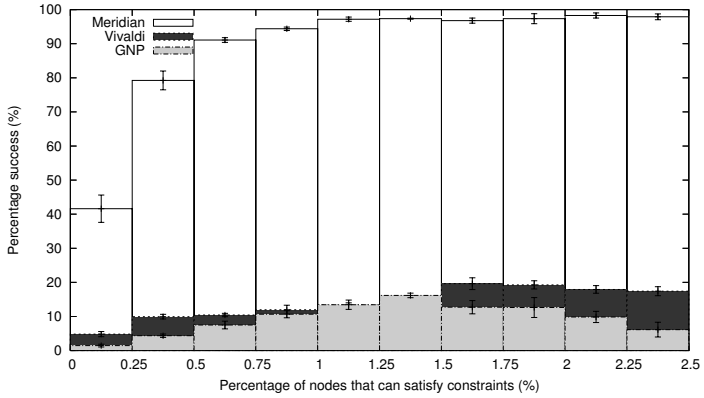


Figure 2.16: The percentage of successful multi-constraint queries is above 90% when the number of nodes that can satisfy the constraints is 0.5% or more.

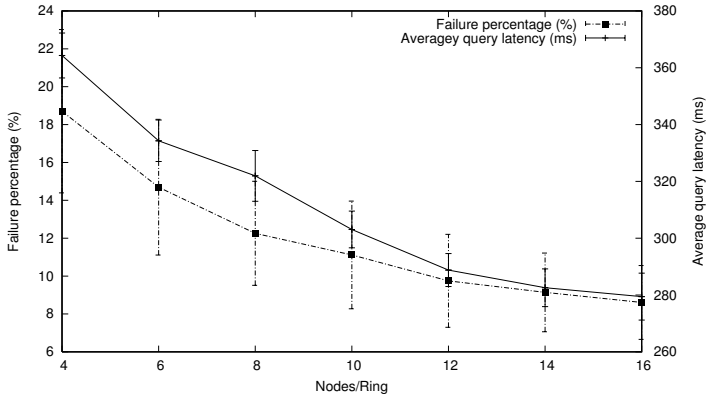


Figure 2.17: An increase in the number of nodes per ring k significantly reduces the failure percentage of multi-constraint queries for $k \leq 8$.

more time cannot be a closer node and its result is discarded. The average query latency curve in Figure 2.10 shows that queries are resolved quickly regardless of k. Average query latency is determined by the hop count and the slowest node in each ring, subject to the maximum latency bound; both increase only marginally as k increases from four to sixteen.

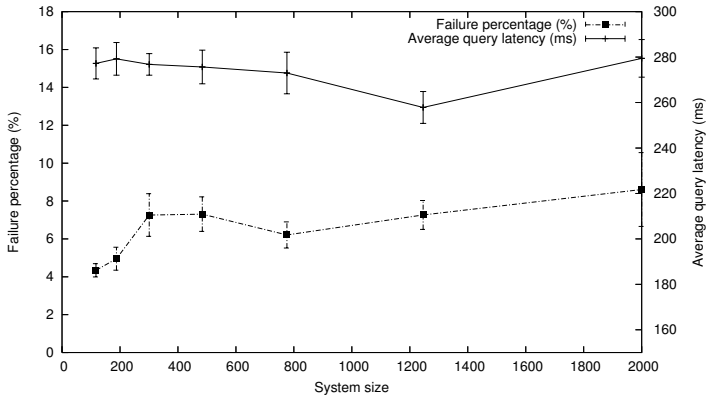


Figure 2.18: The percentage of multi-constraint queries that cannot be resolved with Meridian and average query latency. Both are independent of system size.

The β parameter captures the tradeoff between query latency and accuracy as shown in Figure 2.11. Increasing β increases the query latency, as it reduces the improvements necessary before taking a hop and therefore increases the number hops taken in a query. However, increasing β also provides a significant increase in accuracy for $\beta \leq 0.5$. Accuracy is not sensitive to β for $\beta > 0.5$.

We examine the scalability of the closest node discovery application by evaluating the error, latency and aggregate load at different system sizes. Figure 2.12 plots the median error and average query latency. We set $k = \lfloor \log_{1.6} N \rfloor$ such that the number of nodes per ring varies with the system size; setting k to a constant would favor small system sizes, and this particular log base yields $k = 16$ for 2000 nodes. The median error remains constant as the network grows, varying only within the error margin. The error improves for really small networks where it is feasible to test all possible nodes for proximity. Similarly, the query latency remains constant for all tested system sizes.

Scalability also depends on the aggregate load the system places on the net-

work, as this can limit the number of concurrent closest node discoveries that can be performed at a particular system size. Figure 2.13 plots the total bandwidth required throughout the entire network to resolve a query, that is, the total number of bytes from every packet associated with the query. It shows the results for both the reference implementation with no caching, where a node may naively repeat probes to the target from a single query, as well as an implementation with minimal caching, where nodes retain latency information obtained from probes for the query duration. Both implementations show sublinear growth in average load with system size, with 2000 nodes requiring a total of 10.3 KB and 4.6 KB per query for the uncached and cached implementation. The number of probes per query, a measure of the load imposed on the target, follows the same growth trend as aggregate load, with an average of 53 and 24 probes per query at 2000 nodes for the uncached and cached implementation. The cached implementation requires fewer average probes than Vivaldi, which by default issues 32 probes to compute the coordinates of a new target.

A desirable property for load-balancing is stochastic independence of the ring sets. We verify this property indirectly by measuring the *in-degree ratio* of the nodes in the system. The in-degree ratio is defined as the number of incoming links to a node A over the average number of incoming links to nodes within a ball of radius r around A. If the ring sets are independent, then the in-degree ratio should be close to one; a ratio of one indicates that links to the region bounded by radius r around A are distributed uniformly across the nodes in the area. Figure 2.14 shows that Meridian distributes load evenly. More than 90% of the balls have an in-degree ratio less than two for balls of radius 20ms and 50ms.

Another useful property is that ring members are well distributed. To determine the effectiveness of Meridian's ring membership management protocol, we examine the *latency ratio* of the nodes. The latency ratio for a node A and a target node B is defined as the latency of node C to B over the latency of A to B, where C is the neighbor of A that is closest to B. We find that, for $\beta = \frac{1}{2}$, further progress can be made via an extra hop to a closer node more than 80% of the time. For $\beta = 0.9$, an extra hop can be taken over 97% of the time. This indicates that the ring membership management protocol selects a useful and diverse set of ring members. Compared to a random replacement protocol, we find that the standard deviation of relative error is 38ms when using hypervolumes for selection and 151ms when using random replacement; hypervolume-based selection is more consistent and robust.

We evaluate how Meridian performs in central leader election by measuring its relative error as a function of group size. Figure 2.15 shows that, as group size gets larger, the relative error of the central leader election application drops. Intuitively, this is because the larger group sizes increase the number of nodes eligible to serve as a well-situated leader, and simplify the task of routing the query to a suitable node. Central leader election based on virtual coordinates incurs significantly higher relative error than Meridian for a group size of two. The accuracy gap between coordinate schemes and Meridian closes as the group size increases, as large groups simplify the problem and even random selection becomes competitive with more accurate selection.

We evaluate our multi-constraint protocol by the percentage of queries that it can satisfy, parameterized by the difficulty of the set of constraints. For each multi-constraint query we select four random target nodes and assign a con-

straint to each target node chosen uniformly at random between 40 and 80 ms. The difficulty of a set of constraints is determined by the number of nodes in the system that can satisfy them. The fewer the nodes that can satisfy the set of constraints, the more difficult is the query.

Figure 2.16 shows a histogram of the success rate broken down by the percentage of nodes in the system that can satisfy the set of constraints. For queries that can be satisfied by 0.5% of the nodes in the system or more, the success rate is over 90% for Meridian and less than 11% when using coordinate schemes.

As in closest node discovery, k, the number of nodes per ring, has the largest influence on the performance of the multi-constraint protocol. Figure 2.17 shows that the failure rate decreases as the number of nodes per ring increases. It also shows a decrease in average query latency as the number of nodes per ring increases. An increase in β decreases the failure percentage and increases the average latency of a multi-constraint query, though the performance of the multi-constraint protocol is mostly independent of β.

The scalability properties of the multi-constraint system are very similar to the scalability of closest node discovery. Figure 2.18 shows that the failure rate and the average query latency are independent of system size. The average load per multi-constraint query (not shown) grows sub-linearly and is approximately four times the average load of closest node discovery query. The non-increasing failure rate and the sub-linear growth of the query load make the multi-constraint protocol highly scalable.

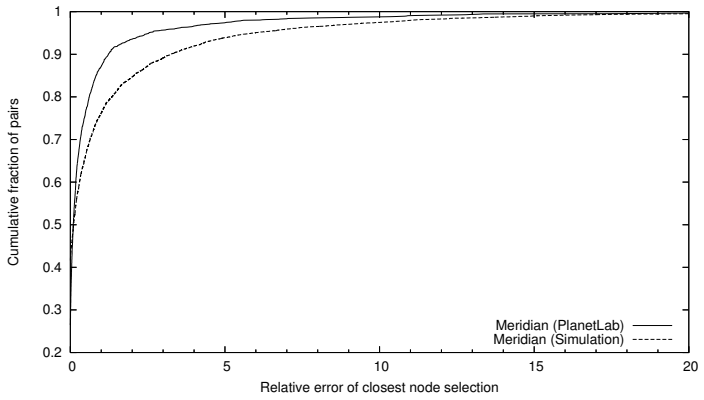


Figure 2.19: The relative error of closest node discovery for a Meridian deployment on PlanetLab versus simulation. Meridian achieves results comparable to or better than our simulations in a real-world deployment.

2.5.2 Physical Deployment

We have implemented and deployed the Meridian framework and all three applications on PlanetLab. The implementation is small, compact and straightforward; it consists of approximately 6500 lines of C++ code. Most of the complexity stems from support for firewalled hosts.

Hosts behind firewalls and NATs are very common on the Internet, and a system must support them if it expects large-scale deployment over uncontrolled, heterogeneous hosts. Meridian supports such hosts by pairing each firewalled host with a fully accessible peer, and connecting the pair via a persistent TCP connection. Messages bound for the firewalled host are routed through its fully accessible peer. A ping, which would ordinarily be sent as a direct UDP packet or a TCP connect request, is sent to the proxy node instead, which forwards it to the destination, which then performs the ping to the originating node and reports the result. A node whose proxy fails is considered to have

failed, and must join the network from scratch to acquire a new proxy. Since a firewalled host cannot directly or indirectly ping another firewalled host, firewalled hosts are excluded from ring membership on other firewalled hosts, but included on fully-accessible nodes.

A large overlay network that performs active probes can potentially be used as a platform for launching denial-of-service attacks. This problem can be avoided either by controlling the set of clients that may inject queries via authentication, or by placing limits on the probing frequency of the overlay nodes. Our implementation chooses the latter and caches the result of latency probes. This considerably reduces the load the overlay nodes can place on a target, as each overlay node can only be coerced to send at most one probe per target within a cache timeout.

We deployed the Meridian implementation over 166 PlanetLab nodes. We benchmark the system with 1600 target web servers drawn randomly from the Yahoo web directory, and examine the latency to the target from the node selected by Meridian versus the optimal obtained by querying every node. Meridian was configured with $k = 8$, $s = 2$, $\beta = \frac{1}{2}$, and $\alpha = 1$ms. Overall, median error in Meridian is 0.54ms, average query latency is 363ms, and the relative error CDF in Figure 2.19 shows that the system performs better than simulation results from a similarly configured system.

2.5.3 Application Performance

The previous results show the improvements in node selection accuracy from using Meridian. In this section, we evaluate the end-to-end improvement in

performance of a distributed application that uses ClosestNode.com, the Meridian to DNS gateway described in Section 2.4, for server selection. We modified the CobWeb service [89], a distributed web-proxy deployed on 484 globally distributed PlanetLab nodes, to start a Meridian daemon process in its node startup scripts and registered the *cob-web.org* domain with the ClosestNode.com DNS server [3]. In addition, we registered the domain *d.cob-web.org* that returns four random CobWeb servers for each request with ten minute TTL values.

We evaluated CobWeb's end-to-end performance by running ApacheBench [95], a standard benchmarking tool for webservers, on the CobWeb deployement. We used Alexa's top 100 most popular websites as the test corpus, and ran ApacheBench against CobWeb on each of the 484 PlanetLab nodes that CobWeb was deployed on. A benchmark run on a node consisted of fetching each of the Alexa sites 10 times, with the median fetch latency for each site being used. Before each benchmark run, we stopped the CobWeb and Meridian processes on the test machine to ensure that the pages were not served locally, and primed the cache of the site's DNS resolver to replicate the performance of the system at steady-state. We ran the benchmark with both Meridian node selection and random node selection. Due to the use of a live service in our experiments, evaluating the performance of CobWeb against other node-selection techniques was not possible as it would have introduced further disruptions to the service. The purpose of this experiment is, therefore, limited to showing the impact of latency sensitive node-selection on the end-to-end performance of a real distributed application.

Figure 2.20 are the CDFs of page fetch latencies from the experiment. It

[3] A service is typically registered as *service.closestnode.com*. In the case of CobWeb, ClosestNode.com's DNS server also served as CobWeb's authoritative name server, allowing the use of the domain *cob-web.org* directly.

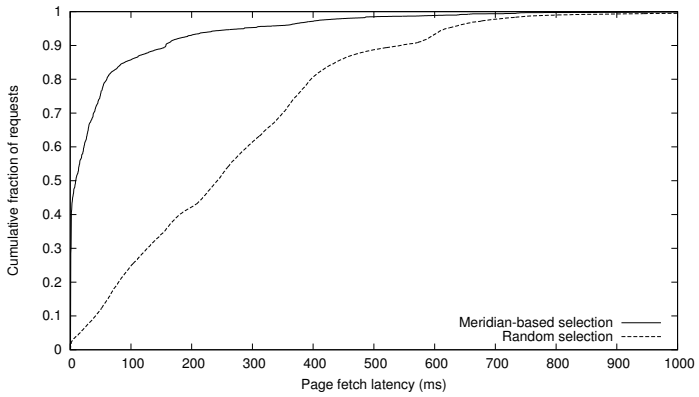


Figure 2.20: The page fetch latency for fetching Alexa's top 100 websites through a distributed web-proxy from 484 different PlanetLab nodes.

shows a substantial reduction in page fetch latency from using Meridian. CobWeb's median page-fetch latency was reduced from 242 ms to 9 ms by using Meridian node-selection in place of random selection. The 95th percentile page-fetch latency was similarily reduced from 619 ms to 287 ms. These results show that Meridian's closest node discovery protocol is able to significantly reduce the page-fetch time of CobWeb, and that latency-sensitive node-selection is a critical element in building a high-performance distributed service.

2.6 Summary

Selecting nodes based on their network location is a critical building block for many large scale distributed applications. Network coordinate systems, coupled with a scalable node selection substrate, may provide one possible approach to solving such problems. However, the generality of absolute coordinate systems comes at the expense of accuracy and complexity.

In this chapter, we outlined a lightweight, accurate and scalable framework for solving positioning problems without the use of explicit network coordinates. Our approach is based on a loosely structured overlay network and uses direct measurements instead of virtual coordinates to perform location-aware query routing without incurring either the complexity, overhead or inaccuracy of an embedding into an absolute coordinate system or the complexity of a geographic peer-to-peer routing substrate.

We have evaluated our system through a PlanetLab deployment as well as extensive simulations, parameterized by data from measurements of 2500 nodes and 6.25 million node pairs. The evaluation indicates that Meridian is effective; it incurs less error than systems based on an absolute embedding, is decentralized, requires relatively modest state and processing, and locates nodes quickly. We have deployed a DNS to Meridian gateway that enables oblivious clients to issue Meridian lookups, reducing the amount of work required to incorporate Meridian into other systems. We have shown how the framework can be used to solve three network positioning problems frequently-encountered in distributed systems, and described a domain specific language that can be used to express other application-specific algorithms. We will show in the next chapter how the lightweight approach we used for performing location-aware node selection can be applied to solving problems in object localization.

CHAPTER 3

APPROXIMATE MATCHING FOR PEER-TO-PEER OVERLAYS

Applications of information retrieval are pervasive in today's computing landscape. Most users navigate the web exclusively via Google or Bing, and large corporations such as Walmart keep petabytes of customer data that are exhaustively mined to help them make informed business decisions. These applications typically rely on large centralized databases that provide good performance and support complex queries, yet such databases are difficult and expensive to operate, placing them out of reach of applications without significant financial rewards.

Peer-to-peer data distribution and retrieval techniques have been tasked to serve these less financially rewarding applications, and have become widely deployed because of their cost-effectiveness, scalability, and resilience to attacks. Yet imprecision stemming from partial specifications of keywords, common variations of search terms, and misspellings pose significant problems to locating content in a peer-to-peer system. Efficiently routing a query to a set of objects whose keys are close but not identical to the search key is a difficult problem known as *approximate matching*.

In this chapter, we present Cubit, a scalable peer-to-peer system that can efficiently find the k closest data items for any search key. It is, to our knowledge, the only efficient peer-to-peer overlay that provides an approximate match primitive. We describe a new technique for (conceptually) mapping nodes and object into a *keyword space*, a metric space that captures the similarity of keywords, and provide algorithms to efficiently route within this space. Finally, we present results from both a real deployment and large-scale simulations that

shows the system is accurate, efficient, and robust. In particular, it can place the target object in the top 20 results for more than 94% of the queries even with a high degree of perturbation in the search terms.

3.1 Approach

An object stored in Cubit is characterized by one or more *keywords*. Cubit's approach to approximate matching relies on an accurate notion of distance between keywords. Such distance should correspond to the users' intuition on which keywords are similar and which are different. The choice of any particular distance is driven by domain requirements; the Cubit's core is agnostic to this choice. For example, Euclidean distance would be a reasonable choice if a keyword is a vector of network coordinates of a node (e.g. [69]), whereas relative entropy would be more appropriate for representing the distance between the feature vectors of two images (e.g. [67]).

3.1.1 Keyword Space

In this chapter, we focus on keywords that are (short) text strings, such as artist names or words in a movie title. Our notion of distance targets misspellings; in particular, the distance between a given keyword and its misspelling should be small. Cubit mainly uses the most common notion of distance on strings, the *Levenshtein distance*, commonly known as the *edit distance*. It is equal to the minimum number of insertions, deletions and substitutions needed to transform one string to another. We also evaluate Cubit using the *Damerau-Levenshtein distance*, an extension of the edit distance that includes the transposition of two charac-

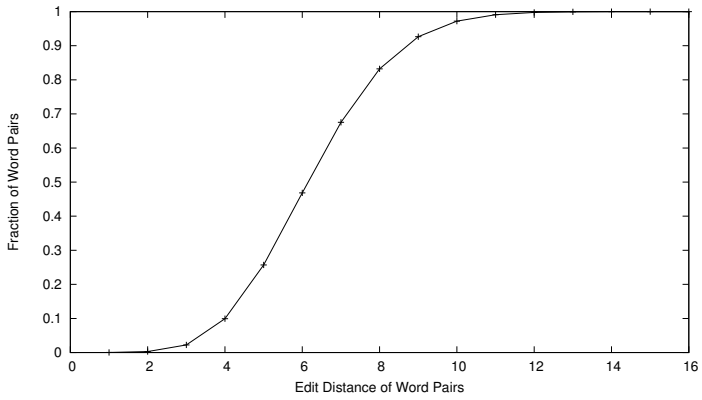


Figure 3.1: The edit distance between pairs of keywords in the Netflix data set: most distances are very small.

ters (a common typographical error) as a single operation. Once the notion of distance is fixed – throughout the chapter it will be the edit distance, unless noted otherwise – the keywords intrinsically lie in the *keyword space*, a metric space on keywords with a metric given by the edit distance.[1]

Consider a typical keyword space taken from the movie database released by Netflix [68] consisting of about $12,000$ keywords from $17,770$ movie titles. By definition, all edit distances are integer values. Since most keywords are short, distances in the keyword space tend to be small (see Figure 3.1). Thus the size of a ball around a typical node grows with the radius much faster than (say) in a two-dimensional grid. In fact, a typical keyword space is very different from the "standard" metric spaces such as Euclidean space. To appreciate this difference, consider the example in Figure 3.2 with a set of five keywords which cannot be embedded into the coordinate plane. Such an embedding becomes increasingly more inaccurate with additional keywords, even if we allow more dimensions. Cubit's use of the keyword space obviates such an inaccurate embedding.

[1] A *metric space* on a set X is a pair (X, σ), where σ is a *metric*, i.e. a non-negative symmetric function σ that obeys $(\sigma(a,b) = 0 \iff a = b)$ and triangle inequality $\sigma(a,c) \leq \sigma(a,b) + \sigma(b,c)$.

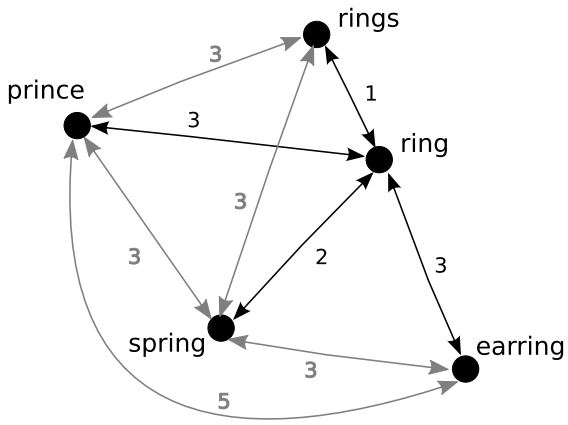


Figure 3.2: The edit distance between keywords: five keywords which cannot be embedded into the coordinate plane so that all distances are preserved. Preserving the distances between four nodes (all but *ring*) distorts the distances to the fifth node.

3.1.2 Multi-Keyword Matching

Search queries typically consist of more than one keyword. Moreover, not all keywords relevant to the desired object may be included in the query, and some of the keywords may be misspelled. For example, a user may search for a long movie title using only a few (misspelled) keywords among those that appear in the title. Cubit matches multi-keyword queries to objects using the *phrase distance* between a query and an object, which we define as the sum, over all search terms, of the minimal edit distance between the search term and the object's keywords. Note that the ordering of keywords does not matter.

Cubit also supports user-specified boolean operators, specifically AND, OR and NOT, between keywords and phrases to enable the construction of expressive, multi-keyword search queries. An object keyword matches a term in the boolean expression if it is within a threshold distance to the search key in the

keyword space. The returned results consist of the set of objects that evaluate to true to the boolean expression.

3.1.3 Node ID Assignment

Cubit nodes are distributed in the same space as keywords. Each node in Cubit is assigned a unique string ID, which determines its "position" in the keyword space. The position determines how a given node is used in Cubit. First, each Cubit node is responsible for storing the set of keywords for which it is the closest node. Second, Cubit implements a distributed protocol which navigates through nodes in the keyword space and locates nodes that store possible matches based on their positions. The details of the protocol are not critical at this stage; the crucial point is that the navigation happens within the keyword space rather than on a ring or some other highly structured artificial routing space of a typical structured peer-to-peer network.

Node IDs are chosen to provide a good coverage of the keyword space. A natural approach is to choose node IDs at random. Since the distribution of words in a human language is known to be very different from that of random strings, we instead choose node IDs at random *among keywords* associated with previously inserted objects in the system. Specifically, at join time each node independently selects a random keyword from those that have been inserted into the overlay, ensuring uniqueness by detecting ID collisions.

3.1.4 Navigation

The navigation protocol is the core component of Cubit. To support this protocol, Cubit creates and maintains a multi-resolution overlay network on nodes such that each node has several peers at every distance from itself; the peers at a given distance are chosen to maximize the coverage of that region. Such overlay design is inspired by the small-world construction [50] in which a grid is augmented by a sparse set of randomly chosen edges, with roughly the same number of edges for each distance scale. In the resulting graph, a greedy routing algorithm (which on each step minimizes the distance to target) succeeds in finding short routes to any given target with high probability.

In Cubit, the distance scales are linear rather than exponential because the keyword space has a very small diameter. The small-world-like overlay is created via an underlying low-overhead gossiping protocol under which nodes randomly exchange peer identifiers and thus randomize their peer sets. Since the distance to the target can be easily computed from the corresponding node ID, the greedy routing algorithm requires very little state and is easy to implement in practice. Both the overlay creation and the small-world navigation happen, essentially, in the keyword space.

3.2 Framework

The basic Cubit routing framework builds on the small world overlay introduced in Chapter 2 for routing in the network latency space. The framework relies on multi-resolution rings to organize peers, a ring membership replacement

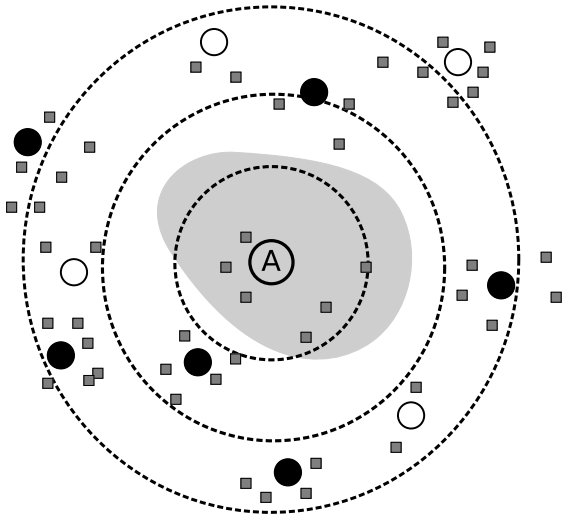


Figure 3.3: A Cubit node organizes its peers into concentric rings, each with a fixed number of nodes. In this example, the solid circles represent peers in node *A*'s peer-set, the empty circles represent other nodes, and the squares represent object keywords in the system. The shaded region depicts the sub-space that is closer to *A* than any other node. The master record for each keyword in the shaded region is stored at node *A*.

scheme to maximize the usefulness of ring members, and a gossip protocol for node discovery and membership dissemination.

3.2.1 Multi-Resolution Rings

Each Cubit node organizes its peers into a set of concentric rings. In each ring, a node retains a fixed number, k_{ring}, of neighbors whose distance to the host lies within the ring boundaries. This ring structure enables a Cubit node to retain a relatively large number of pointers to other nodes within its vicinity, while also providing a sufficient number of pointers to far-away peers.

The Cubit ring structure is illustrated in Figure 3.3. The ith ring has inner radius $R_i = \alpha i$ and outer radius R_{i+1}, for $i \geq 0$, where α is a constant. (We use $\alpha = 1$.) Each node keeps track of a finite number of rings; all rings $i > i^*$ for a system-wide constant i^* are collapsed into a single, outermost ring that spans the range $[\alpha i^*, \infty]$.

In addition to the multi-resolution rings, each node maintains a small *leaf set*, a set of nodes used for object replication management and collision detection on node joins. The leaf-set contains a node's (βf_{repl})-closest neighbors, where $\beta \geq 1$ is a parameter and f_{repl} is the *replication factor*; that is, the number of nodes at which each keyword is replicated.

3.2.2 Ring Membership Management

The number of nodes per ring, k_{ring}, represents a trade-off between accuracy and overhead. A large value of k_{ring} allows each node to retain more information for better route selection during query routing, but requires additional overhead in both memory and bandwidth. The utility of a ring member is in relationship to the amount of diversity it can provide to the ring. Diverse ring members provide better coverage and minimize "holes" in the keyword space, reducing the likelihood that a node is overlooked in query routing.

For each ring, the node retains a fixed number l_{ring} of additional nodes that serve as potential ring candidates. During ring membership selection, an infrequent periodic event, the node selects a subset of k_{ring} ring members from the $k_{\text{ring}} + l_{\text{ring}}$ candidates. The goal is to achieve a good coverage of the corresponding annulus in the keyword space. The specific heuristic used to ac-

complish this is to assign each candidate node a point in the (k_{ring} + l_{ring})-dimensional space, where each dimension represents its distance to one of the candidate nodes, and choose a subset of k_{ring} nodes that forms a polytope with the largest hypervolume. Any heuristic for picking a geometrically diverse set of peers would suffice; the polytope volume provides a principled way to select such diverse peers.

3.2.3 Gossip Based Node Discovery

Cubit uses a standard anti-entropy push-pull protocol [29] for node discovery and dissemination. At each gossip round, a Cubit node collects a random selection of its ring members, and pushes this collection along with its own node information to a random member in each of its rings. At the same time, it pulls back a random selection of nodes from each of the selected ring members. The exchanged nodes are kept as members in the appropriate ring or as replacement candidates if the ring is full. Additionally, nodes exchange their leaf-set with their leaf-set members periodically at a more frequent rate. This ensures that changes to the leaf-set are disseminated more quickly than changes to more distant neighbors.

3.2.4 Replication Management

Cubit replicates objects in order to achieve high availability. The number of replicas of an object naturally falls over time as nodes exit the system. We introduce a simple replication management protocol to maintain the number of

Algorithm 1: MAINTENANCE PROTOCOL

```
          E: Timeout event    R: Local ring set
Require:  L: Local node       O: Object repository
          H: Leaf set         Y: Replication factor
 1: if E.TYPE() = GossipTimer then
 2:     W ← R.SELECTRANDOMNODES()
 3:     for all N in W do
 4:         N.SEND(GossipRequest, (W - N + L))
 5:     W ← H.GETNODES()
 6:     for all N in H.GETNODES() do
 7:         N.SEND(GossipRequest, (H.GETNODES() - N + L))
 8: else if E.TYPE() = ReplacementTimer then
 9:     D ← R.GETRANDOMRINGINDEX()
10:     A ← R.GETPRIMARY(D) + R.GETSECONDARY(D)
11:     B ← {}
12:     while A.LENGTH() > R.MAXNODESPERRING() do
13:         M, V ← NIL, 0
14:         for all N in A do
15:             S ← POLYTOPEVOLUME(A - N)
16:             if M = NIL or S > V then
17:                 M, V ← N, S
18:         A, B ← A - M, B + M
19:     R.SETRING(D, A, B) {Set A to primary, B to secondary}
20: else if E.TYPE() = ReplicaTimer then
21:     M ← O.GETALLMASTERREPLICAS(H.GETNODES())
22:     for all C in M do
23:         for all N in H.GETCLOSEST(Y-1, C) do
24:             N.SEND(CheckKeyRequest, C)
25: else if E.TYPE() = CheckPrimaryTimer then
26:     M ← O.GETALLSECONDARYREPLICAS(H.GETNODES())
27:     for all C in M do
28:         for all N in H.GETCLOSEST(1, C) do
29:             N.SEND(CheckKeyRequest, C)
```

replicas at the desired level f_{repl}.

The *primary node* for a given keyword is the one closest to the keyword, with a fixed tie-breaking rule. This node is responsible for the keyword and its associated objects, and the replication thereof. Each node periodically checks if it is the primary node for the keywords currently at the node. This check can be performed locally by comparing the keywords with the node IDs of the nodes in the leaf-set. It is possible (though unlikely) that for a brief time interval two or more nodes will consider themselves primary for the same keyword. Such

behavior does not reduce accuracy of the search protocol; at worst, it can only *increase* replication level. Each node ensures that an object is replicated at the $f_{\text{repl}} - 1$ closest leaf-set members for each of its keywords that map to that node. Missing replicas are re-created from the primary copy and disseminated to the appropriate nodes. Replicas are reaped locally at the expiry of their leases.

At an even lower periodic rate, each node verifies that, for each of the secondary replicas it owns, the primary node for the replica has a copy of the object. This additional check ensures that the primary node will eventually have a copy of the object if there exists a replica somewhere in the network, limiting the impact of transient routing errors that cause incorrect initial placement of the replicas. Algorithm 1 illustrates Cubit's periodic maintenance operations.

3.2.5 Load Balancing

Since search terms tend to follow a Zipf distribution, the resulting skewed load distribution can lead to excess routing load on nodes within the vicinity of popular keywords. Traditional DHT-based load balancing techniques [75, 26, 79] based on object caching by intermediate nodes are not applicable to Cubit, as an intermediate node can not safely short-circuit a search query unless it can find an exact match. We introduce a novel load-balancing technique that supports short-circuiting of queries for approximate matches.

In Cubit, if the load generated by queries for a popular keyword w overwhelms the available resources of node i, the node manufactures a virtual node at w with all the information it has on that region in the keyword space. This includes the objects in the region and the node's leaf-set, allowing the virtual node

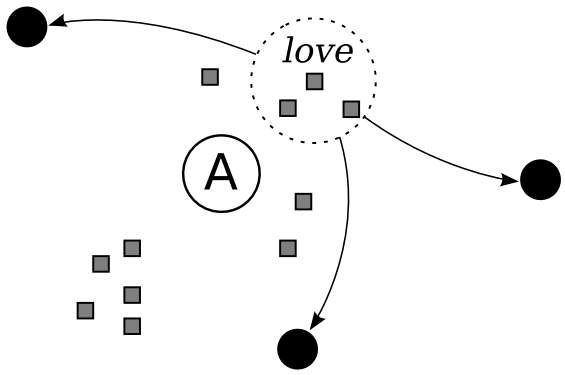


Figure 3.4: Cubit's load-balancing protocol prevents popular keywords from overwhelming a node. In this example, the keyword "love" is closest to node *A* and is generating a high degree of load. Node *A* creates a virtual node around the keyword, which includes its leaf set and all of its objects within a *p* edit-distance radius. This virtual node is sent to *A*'s nearest neighbors to the keyword. Queries that arrive at these neighbors for keywords within the region can be answered without node *A*.

to answer queries on its behalf for that region. The virtual node is disseminated to node *i*'s m_{off} nearest neighbors to *w*, which are the most likely locations to intercept and short-circuit search queries for *w*. Node *i* is then tasked with keeping the m_{off} virtual nodes updated with changes to objects in the off-loaded region as well as changes to its leaf-set. If one of the m_{off} nodes becomes overwhelmed, it can request node *i* to increase the off-loading factor m_{off}. Virtual nodes are not disseminated via gossip and thus do not skew the node distribution. This off-loading operation disperses hot-spots in keyword popularity without requiring global information or coordination. Figure 3.4 illustrates the protocol.

3.3 Query Routing

The following sections describe protocols that use the basic infrastructure described in Section 3.2 to provide the necessary primitives for performing approximate keyword matching.

3.3.1 Object Insert

An object in Cubit is fully described by a set of keywords. In the case of our BitTorrent implementation, these keywords are taken from the filename and embedded comments in the torrent file. The object descriptor is replicated at the r closest nodes to each of its keywords. The form of the object descriptor is unrestricted; in our BitTorrent implementation, a object descriptor is made up of the set of keywords and a pointer to the owner of the torrent file.

When a Cubit node receives an object insertion request, it concurrently issues a closest node search for each keyword using the search protocol described in the next section. The object is initially inserted at the closest node, and the closest node further replicates the object to the f_{repl} - 1 closest neighbors to the keyword, chosen from peers in its leaf-set.

3.3.2 Search Protocol

For non-boolean queries, the goal is to obtain the $k*$ objects nearest to the set of keywords, as measured by the phrase distance, where $k*$ is a parameter in the system. For each keyword in the search phrase, the protocol obtains the $k*$

Algorithm 2: SEARCH PROTOCOL

Require: E: Search event; R: Local ring set; U: Outstanding queries; H: Leaf set

```
 1: N ← E.GETREMOTENODE()
 2: I ← E.GETQUERYID()
 3: K ← E.GETFANOUT()
 4: T ← E.GETKEYWORD()
 5: if E.TYPE() = SearchRequest then
 6:     A ← GETKCLOSESTNODES(T, K, R + H)
 7:     N.SEND(SearchReply, I, T, A)
 8: else if E.TYPE() = SearchReply then
 9:     C ← E.GETRESULTS() - CHECKED[I] - PENDING[I]
10:     CHECKED[I] ← CHECKED[I] + {N}
11:     PENDING[I] ← PENDING[I] + C - {N}
12:     A ← CHECKED[I] + PENDING[I]
13:     A ← GETKCLOSESTNODES(T, K, A)
14:     if A ⊆ CHECKED[I] then
15:         for all V in A do
16:             V.SEND(FetchObjRequest, I, E.SEARCHTERMS())
17:     else
18:         for all V in A ∩ C do
19:             V.SEND(SearchRequest, I, K, D, T)
```

closest objects from each node which meets the following *edit distance criterion*: its ID is within an edit-distance of q from the keyword, where q is the product of the keyword length and the expected number of perturbations per character (which is a parameter in the system). The protocol selects $n_{\min}$ closest nodes if fewer than $n_{\min}$ nodes meet the criterion, where $n_{\min}$ is called the *search fan-out*.

The protocol runs from a fixed node, called the *local node*. It maintains three lists: the *checked list* of nodes that have already been queried, the *pending list* of nodes waiting to be checked, and the *failed list* of nodes such that the corresponding RPC failed or timed out. Initially all three lists are empty.

The protocol inserts the local node into the pending list and enters the following loop. If there exists a node i in the pending list that meets the edit-distance criterion or is equidistant or closer to the keyword than the closest $n_{\min}$

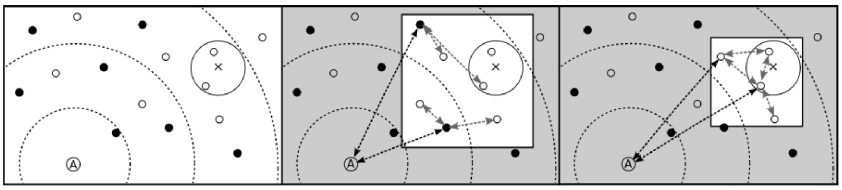


Figure 3.5: The Cubit search protocol iteratively zooms in on the target region. In this example, x is the location of the search term in the keyword space, the solid circles are node A's peers, empty circles are other nodes, and the circle around x includes all nodes within edit-distance q of x. Node A first finds the $n_{\text{min}} = 2$ closest nodes to x from its peer-set, and requests their n_{min} closest nodes. Then two new closer nodes are discovered and subsequently sent the same query. The protocol terminates when all nodes within the circle around x, or the n_{min} closest nodes have been discovered. These nodes are queried for their closest objects to x.

nodes in the checked list, the local node performs an RPC to node i for some of the members in its ring sets: either for all nodes that meet the the edit-distance criterion or for the l_{min} closest neighbors to the keyword, for some constant $l_{\text{min}} \geq n_{\text{min}}$, whichever is larger. If the RPC fails or times out, node i is moved from the pending list to the failed list. Otherwise, it is relocated to the checked list and the new nodes are placed in the pending list unless they have already been checked or have failed a previous RPC. The loop terminates if such a node does not exist.

The $k*$ closest objects to the set of keywords are retrieved either from all checked nodes that meet the edit-distance criterion, or from the n_{min} closest checked nodes, whichever set is larger. The collected objects for all the search terms are ordered by their phrase distance and the $k*$ closest objects are returned as the result of the search. Algorithm 2 shows the pseudo-code for the search

protocol. The edit-distance criterion checks are omitted to improve the clarity and readability of the protocol. Figure 3.5 illustrates an example search query.

3.3.3 Boolean Queries

Advanced search queries in Cubit are composed of boolean expressions, where each term in the expression is either a keyword or a phrase. The search protocol converts a boolean expression into disjunctive normal form, creating a set of conjunction clauses that are connected by OR operators. It uses the standard search primitive to find objects within a threshold phrase distance from each positive term in each clause. The standard search is modified to include all the negative terms in the same conjunction clause that act as filters, ensuring that it avoids returning objects within the threshold distance of these negative terms without requiring explicit searches on them. The union of the results from the conjunction clauses is returned.

Since the number of objects matching a single keyword can be very large, the collection of intermediate results are sent as Bloom filters to reduce the bandwidth requirement of the protocol. The Bloom filters are of sufficient size to distinguish between thousands of objects with a very low (0.1%) false-positive probability and require several orders of magnitude less space than sending the actual objects. The compressed filters are usually only hundreds of bytes in our experiments. Since Bloom filters support union and intersection operations, all intermediate set operations can use the Bloom filters directly. The actual objects that make up the final filter are fetched from the closest nodes of the positive search terms in a final request. In this request, the closest nodes are tasked with

repeating their previous search but would only return objects that are in the final filter.

3.3.4 Node Join

A new node first contacts its given seed nodes to obtain their node IDs and, through a random walk, discovers additional nodes in the network and obtains random keywords from each node. After collecting a sufficient number of nodes, it issues a closest node search for each received keyword. If the closest node's ID is different from the keyword used in the search, then the keyword is used as the node ID for the new node. Simultaneous node joins can, with a very small probability, result in more than one node with the same ID. In this case, the leaf-set discovery will ultimately alert the nodes of the collision, and the node with the lower IP address will drop out and rejoin the system.

Once a unique ID is selected, the new node obtains additional ring members from the ring members of its closest node. It also retrieves the primary replicas of objects with keywords that are closer to the new node than the node they are currently residing at. The protocol for this operates iteratively. It asks each of its k closest nodes if there are any primary replicas that should be copied to the new node that it does not already have. If at least one is closer, the protocol repeats with a larger k until no new primary replicas that should be copied are discovered.

3.3.5 Security

We describe some common attacks targeting the Cubit layer and outline changes to the routing protocol to address them. These changes may incur small performance penalties to query routing. A formal treatment of the security properties of a gossip-based small-world network is beyond the scope of this chapter.

Keyword Hijacking

An attacker can arbitrarily choose as its node ID a keyword for which it wants to return false information. Such information censorship is possible with unmodified Cubit, as the correct execution of the node join protocol cannot be verified by other nodes in the network. To protect against this attack, Cubit can use a node ID selection protocol that deterministically constructs IDs from the IP address of the node. Each node is seeded with the same source of keywords, such as a dictionary, and the hash of the IP address is used as an index into the keywords for selecting the node ID. A remote node's ID is verified before it is added into a node's ring set or before it is used in query routing. This modification primarily affects the distribution of objects across the nodes, so the set of seeded keywords should resemble the keywords in the system. The seeded keywords should at least be taken from the same language as the keywords in the system.

Query Disruption

An attacker can try to disrupt query routing by returning false information. The disruption can be significant in a localized region, prematurely terminating

search and insertion queries. This attack can be circumvented without changes to the existing query protocol by increasing the fan-out factor $n_{\min}$. A query only terminates once the top $n_{\min}$ nodes to the search term is found. By increasing the $n_{\min}$, an attacker has a proportionally smaller influence on query routing in the region. Queries can typically just route around non-cooperating nodes. Increasing $n_{\min}$ comes at a price of additional overhead in query routing. In addition, heavier weight techniques such as PeerReview [42] can be used to identify misbehaving nodes and cleave them from the network.

Spam Injection

An alternative method to disrupt the system is to increase the noise to signal ratio of the keywords and objects in the system. This attack can be addressed in a number of ways. Cubit can provide object insert capabilities only to trusted users by requiring objects to be signed by a certificate authority. Keyword targeted attacks can be bounded by limiting the injection rate. A node can reject an insert request if the same node has been repeatedly inserting the same or similar keyword. A more complete solution is the introduction of a distributed reputation system (e.g. [27]), where poorly rated objects are either discarded or are given a lower rank in response to search queries.

Sybil Attacks

Sybil attacks can allow the attackers to take control of a region of the keyword space. Countermeasures such as [63, 18] can be used to lower the join rate of the attackers, reducing the extent of the attack, or make the attack prohibitively

expensive to undertake, though standard impossibility results apply [30].

3.4 Evaluation

We implemented the full protocol described in the preceding sections as an plugin for the Azureus BitTorrent client. We evaluate Cubit through both a large-scale simulation on real-world datasets and a physical deployment on PlanetLab [11].

3.4.1 Simulation

We use three different real-world datasets to parameterize our simulations. The first is the Netflix database [68], consisting of $17,770$ movie titles. We collected our second dataset by crawling a popular BitTorrent website for media files, consisting of over $39,000$ torrents. These two datasets represent different extremes, with the Netflix dataset providing clean input with no duplicate entries, in contrast to the much noisier BitTorrent data. Our third dataset is the CiteSeer [23] database with the titles of over $400,000$ academic papers. While not representative of file sharing content, the large dataset enables Cubit's sensitivity to the number of objects in the system to be measured at a much broader scale.

The system is evaluated against search queries constructed from keywords of a randomly chosen title, with perturbations introduced to simulate typographical errors and spelling variations. Only two-thirds of the keywords from each title were used in each search query to closer emulate typical user behav-

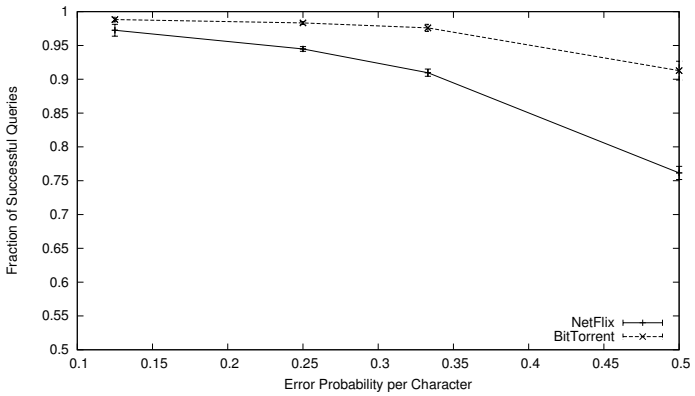


Figure 3.6: Error probability per character vs. accuracy. Queries with high error probability per character are harder to solve as their search terms are very distant from the actual keywords they are derived from in the keyword space.

ior. The error probability per character is the probability that, for each character in the search string, the character has been replaced by a random character. It is a measure of the signal to noise ratio of search keys and is used to control the difficulty of search queries, where a higher error probability represents a more difficult query. Additionally, we evaluated search queries where keywords were modified with real human typographical errors and misspellings from the SearchSpell database [84].

Because of the skewed distribution of English words and the conservative way we are measuring success, 100% success is not always possible. For instance, a query with three typographical errors for "Lost Ark" includes "Last Orc" and might legitimately return an entirely different set of objects; it is possible for the intended object to not be present among the search results if the randomly introduced typographical errors veer into a different portion of the keyword space. We nevertheless retain this conservative success criteria, and

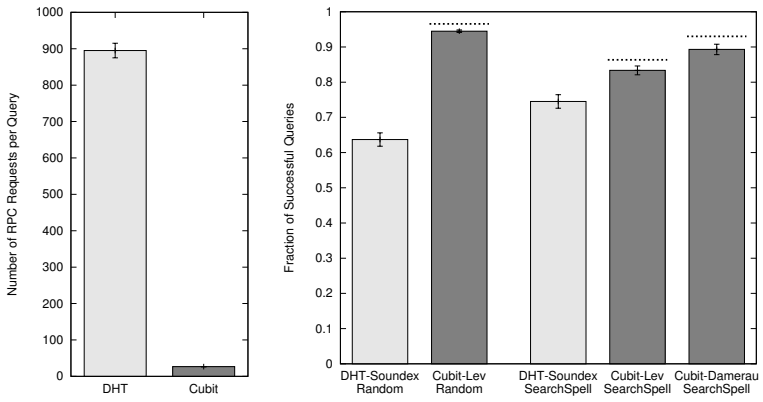
indicate the highest achievable success rate as the *metric upper-bound* where applicable.

In the following experiments, unless specified otherwise, each test consists of 4 runs of 1024 nodes, 10 nodes per ring (88.2 peers per node on average), an error probability per character of 0.25, a search fan-out of 2, a replication factor of 4, with 1000 search queries for each run. The results are presented as the mean result of the runs, and error bars represent 95% confidence intervals. Each simulation run begins from a cold-start, with each new node only knowing at most 8 existing nodes in the network; additional neighbors are discovered through the gossip protocol. An equal fraction of the movies are introduced by each joining node.

Accuracy

We first examine Cubit's accuracy with search queries with increasing levels of difficulty. A search query is considered to be successfully resolved if the original movie it was derived from is a member of the result set, essentially the first page of results presented to the user, which is at most 0.1% of the total number of movies in the system. Our accuracy metric is equivalent to recall, a common statistical classification used in information retrieval [2]. Figure 3.6 shows that Cubit can successfully answer queries where a third of the characters in the search string is expected to be erroneous with more than 90% accuracy. Surprisingly, for queries where half the characters in each search keyword are expected to be perturbed, Cubit is still able to successfully resolve them more than 75% and 90% of the time for the Netflix and BitTorrent datasets respectively. Cubit

[2]One cannot meaningfully present precision, the complementary classification to recall, as our datasets do not include relevance information between objects.



(a) (b)

Figure 3.7: (a) Number of RPC requests per query for a DHT-based system and Cubit. (b) Fraction of successful queries with keywords created from random character perturbations and the SearchSpell database. The dotted lines show the metric upper-bounds.

achieves a higher accuracy on the BitTorrent dataset because the average number of words of a BitTorrent title is 6.6, nearly twice that of a Netflix title at 3.6. A higher number of words per title provides proportionally more keywords per query which improves search accuracy.

The accuracy metric itself does not capture the amount of work and the number of nodes must be contacted to answer the query. A DHT can be 100% accurate if it searches for every misspelled version of a keyword, but would also be highly inefficient. We illustrate the latent costs in Figure 3.7(a). We use a basic DHT implementation based on Pastry [78] for comparison, with a base parameter of 16 and a replication factor of 4. The shortest search term is used by the DHT, as it has the fewest error permutations. For search queries where exactly one error is introduced to each keyword, a DHT solution requires nearly 900 RPC requests before finding the sought object. In contrast, Cubit requires only 27 RPC requests, an order of magnitude fewer than the DHT solution, for

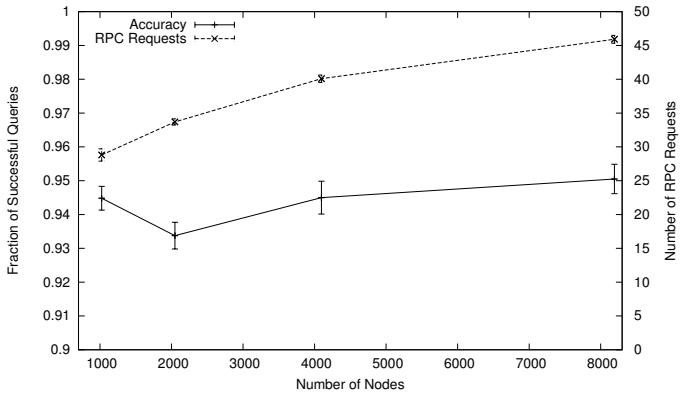


Figure 3.8: Number of nodes vs. accuracy and the number of RPC requests per query.

a query accuracy of more than 96%.

Pairing Soundex hashing, a phonetic algorithm for mapping English words by sound, with DHT routing, as proposed in [105], enables approximate matching without resorting to searching for every possible spelling permutation. Figure 3.7(b) shows that this approach achieves a success rate of only 64% for keywords with random character perturbations and 75% for keywords taken from the SearchSpell database where some of the misspellings are phonetically similar. In contrast, Cubit using the standard Levenshtein distance achieves over 94% accuracy for random character perturbations and 83% accuracy for SearchSpell misspellings. The latter case is due to phonetic misspellings and character transpositions which make up a significant portion of the SearchSpell database do not match well with Levenshtein distance. Even an ideal, centralized service struggles to meet our definition of accuracy when using the Levenshtein distance for matching, with a metric upper-bound accuracy of 97% and 86% for keywords with random character perturbations and SearchSpell misspellings, respectively.

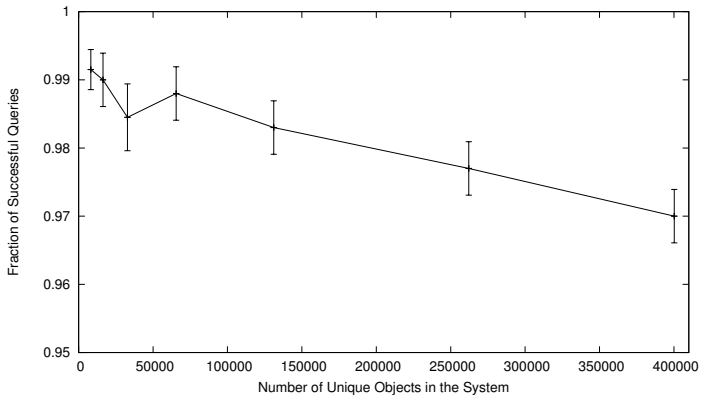


Figure 3.9: Number of objects in the system vs. fraction of successful queries using the CiteSeer dataset.

In contrast, Cubit using the Damerau-Levenshtein distance, which counts the transposition of two characters in a keyword as a single operation, reduces the distance between the misspellings and their origin word and improves Cubit's accuracy to just under 90% for SearchSpell misspellings with a metric upper-bound of 93%. Damerau-Levenshtein based Cubit performs identically with the same overhead in all other experiments as the Levenshtein version; the transposition operation only minimally affects the edit-distance between correctly spelled words, and randomly generated misspellings and their origin word. These results illustrate the importance of using a distance metric that closely matches the expected type of errors, and demonstrate that Cubit is significantly more accurate than Soundex-based solutions for misspellings drawn from both random character permutations and real human typographical errors.

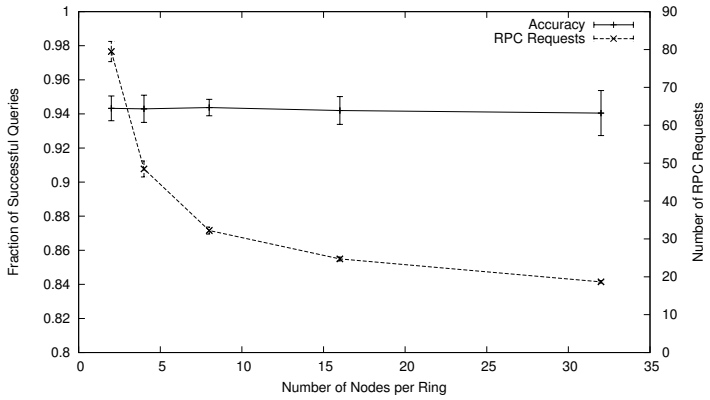


Figure 3.10: Number of nodes per ring vs. accuracy and the number of RPC requests per query.

Scalability

Next, we examine the scalability of the Cubit framework by examining its performance in response to varying system sizes. To be able to directly compare experiments with different numbers of nodes in the network, the number of nodes per ring is configured to be proportional to the logarithm of the system size. Figure 3.8 shows that search accuracy is largely independent of the system size, with Cubit maintaining a 94% accuracy across the tested system sizes. These results indicate that the greedy search protocol very rarely terminates prematurely as a small increase in the hop length due to an increase in system size has a negligible effect on the accuracy. Figure 3.8 also shows how the number of RPC requests per query grows with the number of nodes. The growth rate is proportional to the maximal number of hops times the number of nodes per ring. The former is upper-bounded by a small constant, while the latter is set to log(#nodes) yielding logarithmic scaling.

Another measure of scalability is Cubit's sensitivity to the number of unique

objects in the network. To allow for a more comprehensive evaluation, we use the CiteSeer dataset consisting of more than $400,000$ academic paper titles. In these simulations, rather than returning 0.1% of the total number of unique objects in the system as the result set, we fix the result set to 10 objects to allow for a fair comparison. Figure 3.9 shows the accuracy is minimally affected by an increase in the number of objects in the system. A fifty fold increase in objects results in less than 3% decrease in search accuracy. The search accuracy on the CiteSeer dataset is considerably higher than on the Netflix dataset. This is primarily due to the relatively longer, more distinctive titles found in academic papers, resulting in a sparser, more search friendly keyword space.

A significant concern with boolean search queries is the large number of intermediate search objects that must be sent between nodes. Cubit uses Bloom filters as intermediate storage in order to reduce the bandwidth overhead. To verify the technique's effectiveness, we examine the bandwidth requirement of boolean searches on the Netflix dataset which on average contains 3.6 keywords per object. In our experiments, we use the conjunction of two-thirds the total keywords of the object as the boolean search expression. The median bandwidth used per boolean search query is only 17 KB with only trace amounts of false positives.

System Parameters

The performance of Cubit depends on several key parameters, such as the number of nodes per ring and the query fan-out factor. The number of nodes per ring represents a tradeoff between protocol maintenance and query performance. A low nodes per ring value provides poor coverage of the space and requires more

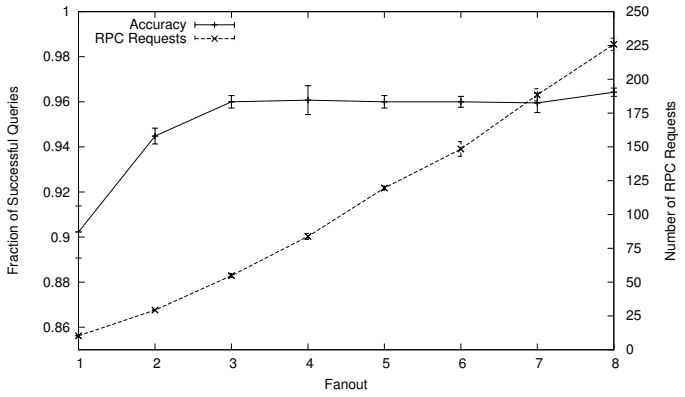


Figure 3.11: Search fanout vs. accuracy and the number of RPC requests per query.

RPC requests to complete a query, where a high nodes per ring value requires additional state to be kept and maintained at each node. Figure 3.10 shows that accuracy is mostly unaffected by the number of nodes per ring, in contrast to the number of RPC per query which decreases dramatically from two nodes per ring to four, and flattening out at sixteen nodes per ring.

The query fan-out bounds the number of closest nodes a query traverses simultaneously, and can significantly improve accuracy by circumventing dead-end paths. For example, a query with a fan-out of two will attempt to find the two closest nodes to the search term at every step, essentially interweaving two simultaneous closest node queries without introducing overlaps in the search space. Figure 3.11 illustrate that increasing fanout from one to two nets a 4% improvement in accuracy, with further increases netting subsequently smaller gains. However, the accuracy comes at the cost of requiring additional RPC requests that increase linearly with the fan-out factor.

The object replication factor also plays a role in the performance of the sys-

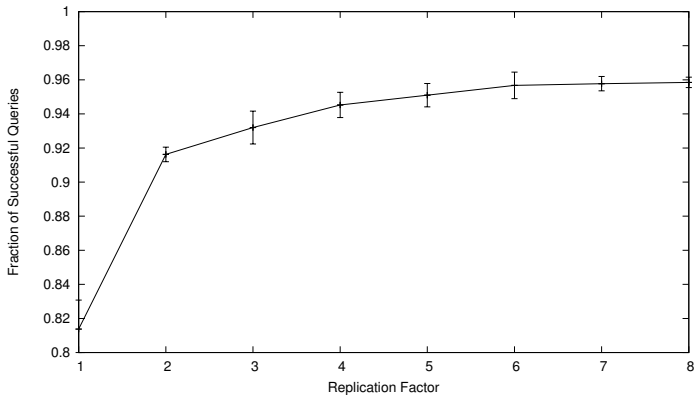


Figure 3.12: Replication vs. accuracy. Modest amounts of replication can yield significant improvement in accuracy.

tem. Figure 3.12 shows that increasing replication from one to four increases search accuracy by more than 12%. Increasing the replication factor beyond four gives only marginal accuracy improvements.

Replication and Churn

The object replication factor trades off accuracy against bandwidth for replica management. Figure 3.12 shows that increasing replication from one to four increases search accuracy by more than 12%. Increasing replication beyond four gives only marginal accuracy improvements. The bandwidth requirement is proportional to the replication factor, the average number of movies per node, and the node churn rate. To quantify the bandwidth requirement for replica management, we added churn to our simulations. The node lifetime distribution was collected from our Azureus deployment of more than $6,000$ Cubit users. Under this realistic churn scenario, the bandwidth required for replica management is less than 1.6 KB/s for each Cubit node, which compares favor-

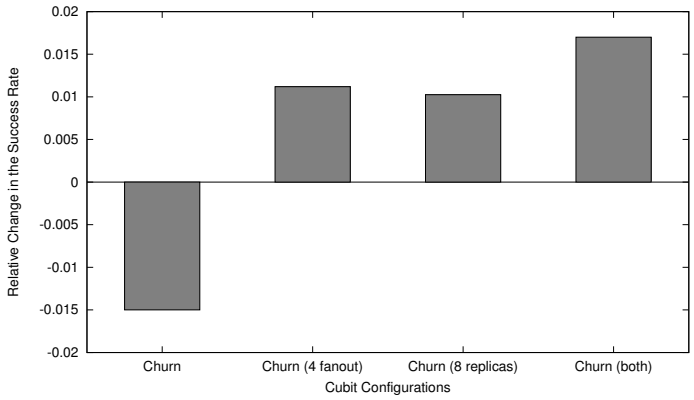


Figure 3.13: Relative change in accuracy due to churn. The first bar shows that realistic churn rates have modest effects. Optimizations, including increased fanout, increased object replication, and both combined, can compensate for the effects of churn.

ably to 2 KB/s used for the maintenance of the BitTorrent DHT on a host with a 32 KB/s upstream connection [14].

Beyond its effect on maintenance traffic, node churn can also negatively affect search accuracy. This is primarily due to stale ring members that create "holes" in the keyword space, preventing queries from routing to the target region. However, introducing node churn into the simulation results in a barely perceptible decrease in search accuracy (Figure 3.13). This is because the gossip rate is sufficiently high to detect and remove stale ring members. In our deployment, an average ring member receives a gossip request every two minutes, and the measured median lifetime of a node is 20 minutes. Raising the values of other system parameters, such as the query fan-out and replication factor, provides ways to maintain search accuracy under even higher levels of churn.

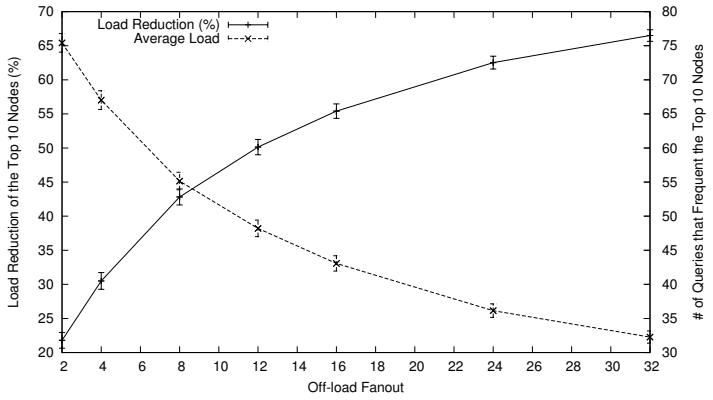


Figure 3.14: Offload fanout vs. load at hotspots. Cubit's load balancing protocol is able to significantly spread the load away from load hotspots.

Load-balancing

We next examine how well the load-balancing protocol disperses hotspots in query routing. In this experiment, we overload the system by issuing a misspelled keyword query from 100 random nodes. In response, the top ten most highly frequented nodes request their neighbors to create virtual nodes. We then repeat the queries and compare the concentration of queries that frequent the top ten most visited nodes before and after virtual node creation. We vary the offload fan-out γ and plot the average number of queries that frequented the top ten nodes and their reduction in average load. Figure 3.14 shows that the Cubit load-balancing protocol is effective at reducing the load at request hotspot through the introduction of virtual nodes. Even an off-load fanout of 8 can reduce the load by more than 40% on average.

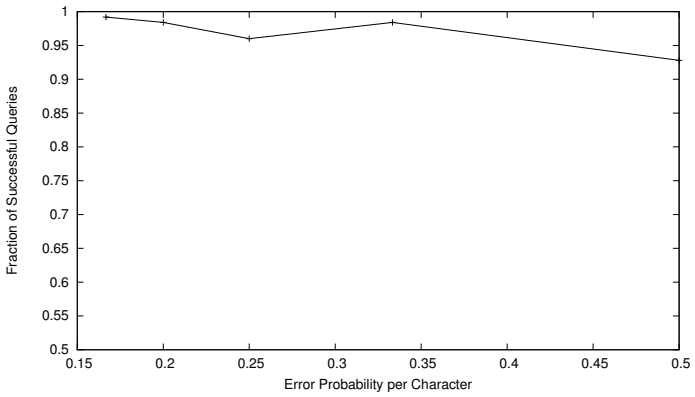


Figure 3.15: Error probability per character vs. the accuracy in the Azureus deployment.

3.4.2 Azureus Deployment

We implemented a Cubit plugin for the Azureus BitTorrent client to provide approximate matching of available torrents. The torrents are currently taken from crawls of popular torrent websites and from trackerless torrents in the Azureus DHT. Torrents in the system automatically expire after a set time-out; persistence beyond a single time-out requires reinjections, similar to trackerless torrents.

The system is currently deployed, with 107 PlanetLab nodes acting as gateway nodes to the network. More than $10,000$ torrents have been injected into the system. We examine Cubit's accuracy on the Azureus deployment by issuing 125 search queries with different error probability per character values. Figure 3.15 shows that Cubit can successfully answer queries where half the characters are expected to be perturbed with more than 90% accuracy which closely matches our simulation predictions. The plugin has been made publicly available at the Cubit project website [99]. There are currently more than $6,000$

active users of the Cubit plugin.

3.5 Summary

This chapter describes Cubit, the first peer-to-peer overlay to provide an efficient approximate match primitive. The key insight behind Cubit is to create a keyword metric space that captures the relative similarity of keywords, to assign portions of this space to nodes in a light-weight overlay and to resolve queries by efficiently routing them through this space, allowing Cubit to quickly identify approximately matching objects to a set of search terms.

Cubit has been implemented as a BitTorrent client plugin with more than $6,000$ active users, and evaluated through a PlanetLab deployment as well as through extensive simulations using large, real-world datasets. The evaluation indicates that Cubit is scalable, accurate, and efficient – it uses an order of magnitude less communication than naive extensions to DHT systems and is significantly more accurate than systems based on Soundex hashing. The technique is immediately applicable to domains, such as peer-to-peer filesharing, where query terms are provided by users and require a decentralized approximate match against objects in the system.

CHAPTER 4

GEOGRAPHIC LOCATION OF INTERNET HOSTS

Determining the physical location of an Internet host is a key enabler for a wide range of network services. Accurately determining the position of a node in the real world based solely on network measurements, however, poses many challenges. The key obstacles to accurate and precise geolocalization are threefold: how to represent network locations for nodes, how to extract constraints on where nodes may or may not be located, and how to combine these constraints to yield good estimates of node position [1].

This chapter presents a novel, comprehensive framework for geolocating Internet hosts called Octant [2]. Octant provides a framework which represents node positions precisely using regions, expresses constraints succinctly as areas, and computes positions accurately by solving a system of geometric constraints. A small number of landmarks whose positions are approximately known anchor the constraint system to the physical globe. The Octant approach is comprehensive and general; it enables almost all past work on geolocalization to be expressed within the framework, as a (limited) subset of the techniques described in this paper.

4.1 Framework

The goal of the Octant framework is to compute a region β_i that comprises the set of points on the surface of the globe where node i might be located. This

[1] In this context, *accuracy* refers to the distance between the computed point estimate and the actual location of the target. In contrast, *precision* refers to the size of the region in which a target is estimated to lie.

[2] An octant is a navigational instrument used to obtain fixes.

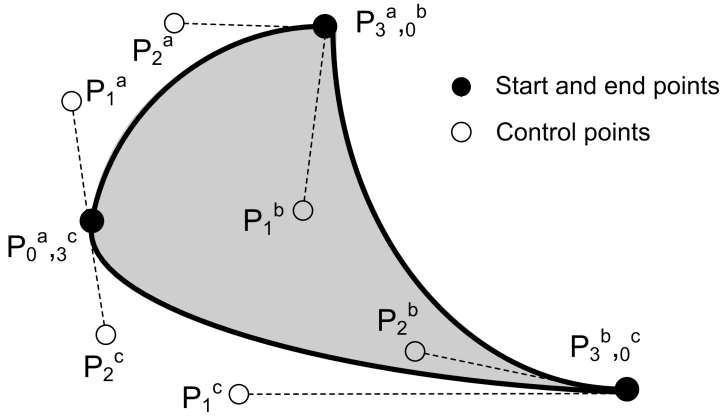


Figure 4.1: Location representation in Octant. Octant represents the estimated target location as a region bounded by a set of Bézier curves. Each curve a, b, c consists of four control points $P_0, ..., P_3$ with P_0 and P_3 as the start and end points respectively and P_1 and P_2 as control points that help direct the curve. This figure requires a total of only nine control points to precisely define. Bézier curves provide a compact way to represent large, complex areas precisely. They also admit efficient intersection, union, and subtraction operations.

estimated location region β_i is computed based on constraints $\gamma_0 \ldots \gamma_n$ provided to Octant.

A constraint γ is a region on the globe in which the target node is believed to reside, along with an associated weight that captures the strength of that belief. The constraint region can have an arbitrary boundary, as in the case of zip-code information extracted from the WHOIS database or coastline information from a geographic database. Octant represents such areas using Bézier-regions, which consist of adjoining piecewise Bézier curves as illustrated in Figure 4.1. Bézier curves are polynomial parametric curves with n+1 control points $P_0, ..., P_n$ where n is the order of the polynomial, with $n = 3$ for most implementations.

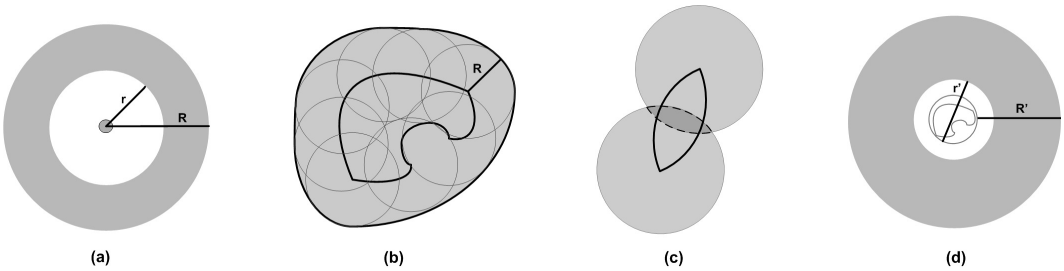


Figure 4.2: Comprehensive use of positive and negative constraints in Octant. (a) A primary landmark, with a precise position estimate, and its associated constraints. (b) Positive constraints are calculated by taking the union of all circles in the estimated area. A node within distance d must reside in the region marked with the dark outer line. Only a subsample of the circles are shown for clarity. (c) Negative constraints are computed by taking the intersection of all circles in the estimated area. A node outside of distance d can not be in the region marked with the dotted line. (d) A secondary landmark, whose position is not known precisely. Note that the associated constraints lead to a larger annulus, due to the conservative, sound way in which Octant combines them. An implementation may replace complex Bézier regions with a bounding circle for efficiency.

Intuitively, the points P_0 and P_n are the start and end points with the remaining points providing the directional information. Bézier regions provide both precise and compact representation of complex shapes. For example, a circle can be represented exactly using four adjoining Bézier curves and a total of twelve control points.

Typically constraints are obtained via network measurements from a set of nodes, called *landmarks*, whose physical locations are at least partially known. Every landmark node L_j has an associated estimated location region β_{L_j}, whose size captures the amount of error in the position estimate for the landmark.

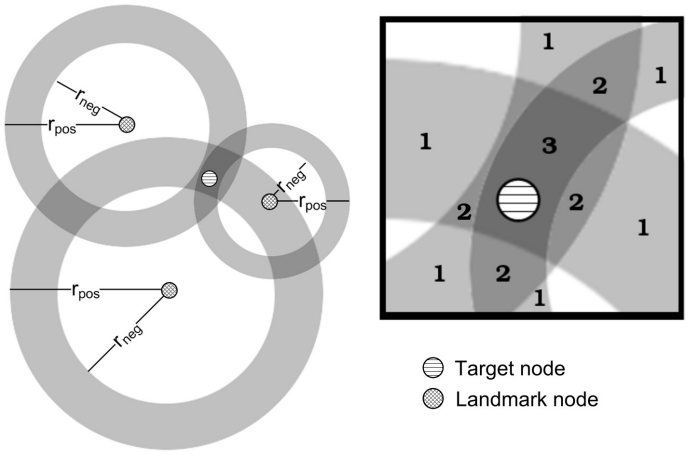


Figure 4.3: Octant computes an estimated location region for a target node by combining positive and negative information available through latency measurements. The resulting location estimate comprises non-convex, potentially disjoint regions separated by weight.

We call a node a *primary landmark* if its position estimate was created via some exogenous mechanism, such as a GPS measurement or by mapping a street address to global coordinates. Typically, primary landmarks have very low error associated with their position estimates. We call a node a *secondary landmark* if its position estimate was computed by Octant itself. In such cases, β_{L_j} is the result of executing Octant with the secondary landmark L_j as the target node.

Octant enables landmarks to introduce constraints about the location of a target node based either on *positive* or *negative* information. A positive constraint is of the form "node A is within x miles of Landmark L_1," whereas a negative constraint is a statement of the form "node A is further than y miles from Landmark L_1." On a finite surface, such as the globe, these two statements both lead to a finite region in which the node is believed to lie. However, the nature of the constraint, either positive or negative, makes a big difference in how these

regions are computed.

In the simple case where the location of a primary landmark is known with pinpoint accuracy, a positive constraint with distance d defines a disk with radius d centered around the landmark in which the node must reside. A negative constraint with distance d' defines the complement, namely, all points on the globe that are not within the disk with radius d'. In typical Octant operation, each landmark L_j contributes both a positive and a negative constraint. When the source landmark is a primary whose position is known accurately, such constraints define an annulus.

Octant enables meaningful extraction of constraint regions even when the position of the landmark is approximate and consists of an irregular region. For a secondary landmark k whose position estimate is β_k, a positive constraint with distance d defines a region that consists of the union of all circles of radius d at all points inside β_k (formally, $\gamma = \bigcup_{(x,y)\in\beta_k} c(x, y, d)$ where $c(x, y, d)$ is the disk with radius d centered at (x, y)). In contrast, a negative constraint rules out the possibility that the target is located at those points that are within distance d regardless of where the landmark might be within β_k (formally, $\gamma = \bigcap_{(x,y)\in\beta_k} c(x, y, d)$).

Given the description above, it may seem that computing these intersection and union regions might take time proportional to the area of β_k, and thus be infeasible. Octant's representation of regions using Bézier curves enables these operations to be performed very efficiently via transformations only on the endpoints of Bézier segments. Since Bézier curves are used heavily in computer graphics, efficient implementations of Bézier clipping and union operations are available. However, the number of Bézier segments in a region increases with each intersection and union operation, which gradually expands the number of

curves to track and manipulate, which in turn poses a limit to the scalability of the framework. So a scalable Octant implementation may decide to approximate certain complex β_k with a simple bounding circle in order to keep the number of curves per region in check and thus gain scalability at the cost of modest error. Figure 4.2 illustrates the derivation of positive and negative constraints from primary and secondary landmarks.

Given a set Ω of positive constraints and a set Φ of negative constraints on the position of a target node i, the estimated location region for the target is given by:

$$\beta_i = \bigcap_{X_i \in \Omega} X_i \setminus \bigcup_{X_i \in \Phi} X_i. \tag{4.1}$$

This equation is precise and lends itself to an efficient and elegant geometric solution. Figure 4.3 illustrates how Octant combines constraints to yield an accurate estimated location region for a target.

There are, however, many issues to solve before this approach can be used for practical geolocalization on the Internet. In the general formulation above, all constraints are weighted equally and the solution is discrete; a point is either part of the solution space or it is not. A discrete solution strategy leads to a brittle system, as a single erroneous constraint will collapse the estimated location region down to the empty set. One strategy is to use only highly conservative positive constraints derived from the speed of light and avoid all negative constraints. We show later that such a conservative strategy does not achieve good precision. In the next set of sections, we detail optimizations that enable the basic Octant framework to be applied to noisy and conflicting measurements on

the Internet.

If latencies on the Internet were directly proportional to distances in the real world, the geolocalization problem would be greatly simplified. Three factors complicate Internet latency measurements. First, the Internet consists of heterogeneous links, hosts and routers whose transmission and processing speeds vary widely. Second, inelastic delays incurred on the last hop can introduce additional latencies that break the correspondence between round trip timings and physical distances. Finally, packets often follow indirect, circuitous paths from a source to a destination, rendering great-circle approximations inaccurate. In the next three sections, we address each of these problems in turn.

4.1.1 Mapping Latencies to Distances

The network latency between a target and a landmark physically bounds their maximum geographical distance. A round-trip latency measurement of d milliseconds between a landmark and a target can be translated into a distance constraint using the propagation delay of light in fiber, approximately $\frac{2}{3}$ the speed of light. This yields a conservative positive constraint on node locations that can then be solved using the Octant framework to yield a sound estimated position for the target; such an estimate will never yield an infeasible ($\emptyset$) solution. In practice, however, such constraints are so loose that they lead to very low precision.

Yet the correlation between latency measurements and real-world distances is typically better and tighter than constraints based on the speed of light. Figure 4.4 plots the network latency against physical distance from a primary land-

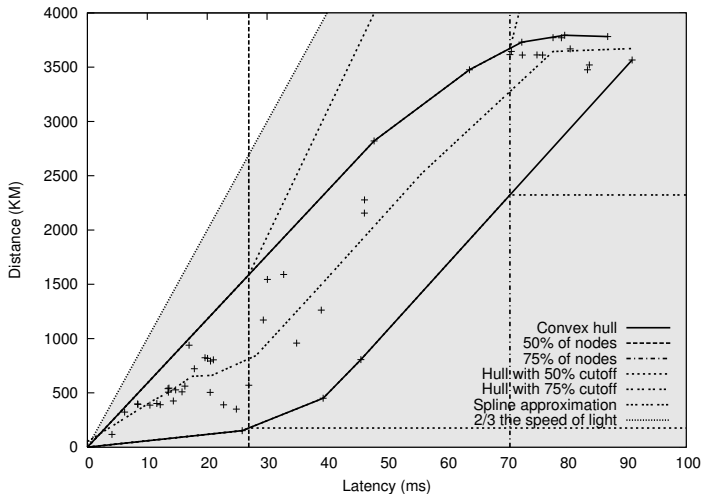


Figure 4.4: The latency-to-distance plot of peer landmarks for a representative landmark (planetlab1.cs.rochester.edu). The shaded region denotes the valid point locations as bounded by the propagation delay time of light in fiber. The convex hull around the data-points serves as the positive and negative constraints for the node. For a given latency, the top and bottom of the hull represent the outer and inner radius respectively of the constraint annulus. As distances increase, fewer representative nodes remain, rendering the convex hull overly aggressive. Vertical lines indicate the 50 and 75th percentile cutoffs, where the convex hull is cut and replaced with conservative positive and negative constraints when insufficient representative nodes remain.

mark (planetlab1.cs.rochester.edu) to all other primary landmarks in our study. The figure makes clear the loose correlation between physical distance and illustrates how overly conservative the speed of light bounds can be. In addition, the empty region to the lower right suggests that few links are significantly congested; nodes that are physically close are typically reachable in a short amount of time. This presents an opportunity for a system wishing to aggressively extract constraints at the risk of occasionally making overly aggressive claims, to

both tighten the bounds on positive constraints and to introduce negative constraints.

Octant calibrates each landmark when the landmark is initialized as well as periodically to determine the correlation between network measurements performed from that landmark and real-world distances. The goal of the calibration step is to compute two bounds $R_L(d)$ and $r_L(d)$ for each landmark L and latency measurement d such that a node i whose ping time is d will be between $r_L(d) \leq ||loc(L) - loc(i)|| \leq R_L(d)$. This permits Octant to extract a positive and a negative constraint for each measurement made from each landmark. Note that when $r_L(d) = 0$, the negative constraint is defunct and does not play a role in localization; for nodes that are so close that ping times are dominated by queuing delays, r_L should be zero.

A principled approach is used to conservatively pick R_L and r_L. Each landmark periodically pings all other landmarks in the system, creating a correlation table much like Figure 4.4. It then determines the convex hull around the points on the graph. Functions R_L and r_L correspond to the upper and lower facets of the convex hull. This approach for extracting constraints is both tight and conservative. The R_L and r_L bounds do not contradict any empirical results, as the convex hull envelopes all data points measured at the landmark. The bounds are significantly tighter than bounds derived from linear functions used in previous techniques [38]. And the convex hull facets are smooth, positively sloped, and closely track the average latency to distance correlation.

In practice, this approach yields good results when there are sufficient landmarks that inter-landmark measurements approximate landmark-to-target measurements. In cases where the target has a direct and congestion-free path

to the landmark, it may lie beyond $R_L(d)$, and vice versa for $r_L(d)$. While extensions to Octant we discuss later can compensate for occasional errors, the r and R estimates may be systematically wrong when there are just insufficient landmarks to draw statistically valid conclusions. Consequently, Octant introduces a cutoff at latency ρ, such that a tunable percentile of landmarks lie to the left of ρ, and discards the part of the convex hull that lies to the right of ρ. That is, only the part of the convex hull for which sufficient data points are available is taken into consideration. Octant then uses $r_L(x) = r_L(\rho), \forall x \geq \rho$, and $R_L(x) = m(x - \rho) + R_L(\rho), m = (y_z - R_L(\rho))/(x_z - \rho)$, where a fictitious sentinel datapoint z, placed far away, provides a smooth transition from the aggressive estimates on the convex hull towards the conservative constraints based on the limits imposed by the speed of light.

4.1.2 Last Hop Delays

Mapping latencies to distances is further complicated by additional queuing, processing, and transmission delays associated with the last hop. For home users, these last hop delays can be attributed to cable and DSL connections that are often under-provisioned. Even in the wide area, the processing overhead on servers adds additional time to latency measurements that can overshadow the transmission delays. For instance, on overloaded Planetlab nodes, measured latencies can be significantly inflated even with careful use of kernel timestamps. Consequently, achieving more accurate and robust localization results requires that we isolate the inelastic delay components which artificially inflate latency measurements and confound the latency to distance correlation.

Ideally, a geolocalization system would query all routers on all inter-node paths, isolate routers that are present on every path from each node, and associate the queuing and transmission delays of these routers along with the node's average processing delay as the inelastic component of the node. Since this approach is impractical, we need a feasible way to approximate the last hop delay from latency measurements.

Three properties of the problem domain motivate an end-to-end approach to the measurement and representation of last hop delay in Octant. First, localization needs to be performed quickly without the cooperation of the target host. This rules out the use of precise timing hardware for packet dilation, as well as software approaches that require pre-installed processing code on the target. Second, creating detailed maps of the underlying physical network, as in network tomography [96, 15], entails significant overhead and does not yet provide answers on the timescales necessary for on-the-fly localization. Third, Octant has mechanisms in place to accommodate uncertainty in constraints (section 4.1.4) and can thus afford imprecision in its last hop delay estimates. These properties led us to use a fast, low-overhead, end-to-end approach for capturing the last hop delay seen on measurements from a given host in a single, simple metric which we call height.

Octant derives height estimates from pair-wise latency measurements between landmarks. Primary landmarks, say a, b, c, measure their latencies, denoted $[a, b]$, $[a, c]$, $[b, c]$. The measure for latency is the round-trip time, which captures the last hop delays in both link directions. Since the positions of primary landmarks are known, the great circle distances between the landmarks can be computed, which yield corresponding estimates of transmission delay,

denoted (a, b), (a, c), (b, c). This provides an estimate of the inelastic delay component between any two landmarks [3]; for instance, the inelastic delay component between landmarks a and b is $[a, b] - (a, b)$. Octant determines how much of the delays can be attributed to each landmark, denoted a', b', c', by solving the following set of equations:

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} a' \\ b' \\ c' \end{bmatrix} = \begin{bmatrix} [a, b] - (a, b) \\ [a, c] - (a, c) \\ [b, c] - (b, c) \end{bmatrix}$$

Similarly, for a target t, Octant can compute t',as well as an estimate of the longitude and latitude, t_{long} and t_{lat}, by solving the following system of equations:

$$\begin{aligned} a' + t' + (a, t) &= [a, t] \\ b' + t' + (b, t) &= [b, t] \\ c' + t' + (c, t) &= [c, t] \end{aligned}$$

where (a, t) can be computed in terms of a_{long}, a_{lat}, t_{long}, t_{lat}. We can then solve for the t', t_{long}, t_{lat} that minimizes the residue. The computed t_{long} and t_{lat} result, similar to the synthetic coordinates assigned by [25], has relatively high error and is not used in the later stages. The target node itself need not participate in the solution for its height, except by responding to pings from landmarks. Figure 4.5 shows the heights of the landmarks in our PlanetLab dataset.

[3]Note that this difference might embody some additional transmission delays stemming from the use of indirect paths. We expand on this in the next section.

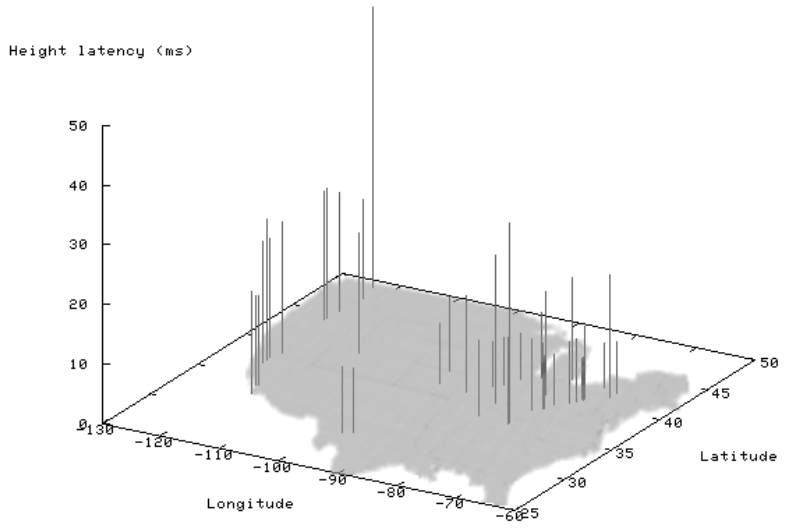


Figure 4.5: Heights computed by Octant to capture last hop delays on network paths to geographically distributed landmarks. Vertical bars represent landmarks, their position corresponds to their physical location, while the length of the bars corresponds to their assigned heights.

Given the target and landmarks' heights, each landmark can shift its R_L up if the target's height is less than the heights of the other landmarks, and similarly shift its r_L down if the target's height is greater than the heights of the other landmarks. This provides a principled basis for ensuring that at least a fraction of the time packets spend in the last hop do not skew the derived constraints.

4.1.3 Indirect Routes

The preceding discussion made the simplifying assumption that route lengths between landmarks and the target are proportional to great circle distances. Studies [80] have shown that this is often not the case in practice, due to policy routing. For instance, routes between subscribers in Ithaca, NY and Cornell University traverse Syracuse, NY, Brockport, IL, and New York City before getting routed back to Cornell, traveling approximately 800 miles to cover less than a mile of physical distance. A geolocalization system with a built-in assumption of proportionality would not be able to achieve good accuracy.

Note that the preceding section on height computation addresses some, but not all, of the inaccuracies stemming from indirect routes. In the example above, if all packets from this landmark get routed through Syracuse, NY, the distance between Ithaca and Syracuse will be folded into the landmark's height, enabling the landmark to accurately compute negative information even for nodes that are near it (without the height, the landmark might preclude its next door neighbors from being located in Ithaca). The height optimization, however, does not address inaccuracies stemming from the inconsistent, or unexpected use of indirect routes. Specifically, nodes with otherwise low heights might choose unexpectedly long and circuitous routes for certain targets. This occurs often enough in practice that accurate geolocalization requires a mechanism to compensate for its effects.

Octant addresses indirect routes by performing piecewise localization, that is localizing routers on the network path from the landmarks to the target serially, using routers localized on previous steps as secondary landmarks. This approach yields much better results than using just end-to-end latencies, and is

illustrated in Figure 4.6. Since Octant can perform localization based solely on round-trip timings, localizing routers does not require any additional code to be deployed.

While the Octant framework can be used as is to locate routers, the structured way in which most routers are named enables Octant to extract more precise information about their location. Octant performs a reverse DNS lookup on each router on the path and determines the city in which it resides by using the **undns** [90] tool, which extracts locations from ISP-specific naming patterns in router names. The city names for routers with city information are converted into geographical coordinates using data from the US census zipcode database. A given city can have multiple coordinates in the database, with each representing the location of a zipcode region. The location of a router of a given city is the bounding circle encompassing the city's coordinates with a tunable slack to account for large zipcode regions. This approach yields much better results than using just end-to-end latencies, as routes between routers separated by a single link is largely void of indirect routing.

4.1.4 Handling Uncertainty

A mechanism to handle and filter out erroneous constraints is critical to maintaining high localization accuracy. The core mechanism Octant uses is to assign weights to constraints based on their inherent accuracy.

For latency-based constraints, we have observed that the accuracy of constraints from landmarks that have high latency to the target are less trustworthy than those that are nearby. The simple intuition behind this is that the in-

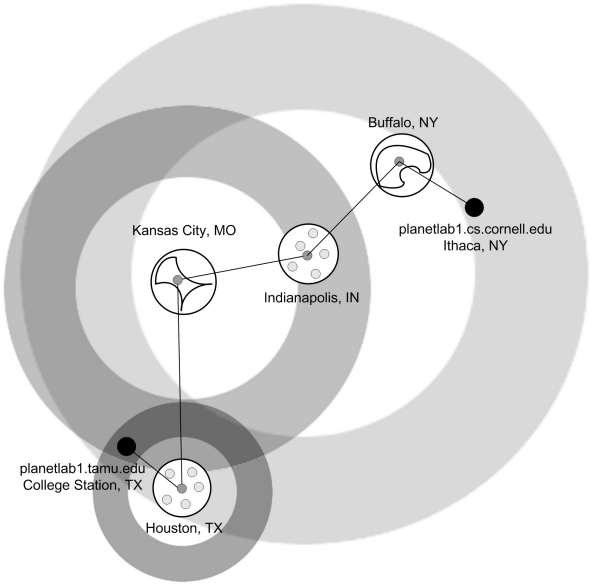


Figure 4.6: The use of intermediate routers as secondary landmarks can significantly increase localization accuracy. Octant is used first to determine the estimated location area for a router. Where possible, Octant refines location estimates based on the city, extracted from the router's DNS name with **undns**, in which the router is located. This is shown for Indianapolis and Houston, where dots represent the center of every zipcode located in that city. For Buffalo and Kansas City, the location of the routers is computed using Octant without undns information. The rings around Buffalo and Ithaca are omitted for clarity.

crease in latency is either due to far-away nodes that have a higher probability of traversing through indirect, meandering routes or travel along paths that have high congestion, which often results in constraints that are of relatively little use compared to nearby nodes.

Octant uses a weight system that decreases exponentially with increasing latency, thereby mitigating the effect of high-latency landmarks when lower latency landmarks are present. A weight is associated with each constraint based

on the latency between the originating landmark and the target node. When two regions overlap, the weights are added together. In the absence of weights, regions can be combined via union and intersection operations, leading to a discrete solution for a location estimate - the node is either within a region, or lies outside. The introduction of weights changes the implementation. When two regions overlap, Octant determines all possible resulting regions via intersections, and assigns the associated weight to each. The final estimated location region is computed by taking the union of all regions, sorted by weight, such that they exceed a desired weight or region size threshold.

Weights enable Octant to integrate constraints of questionable verity into the system. Recall that, when the resulting area is the empty set, going back and finding the minimal set of constraints that led to an infeasible solution is an NP-complete problem. Weights allow conflicting information to be introduced into the system at little risk of over-constraining the final system and reducing its effectiveness; overaggressive constraints from latency measurements, incorrect zipcode from WHOIS, or misnamed routers in **undns** will not just render the solution down to the empty set. Bad constraints may still impact accuracy if there are no compensating factors, but weights enable Octant to associate a probability measure with regions of space in which a node might lie.

4.1.5 Iterative Refinement

Localization in the Octant framework can be broken down into two phases. The first is to use accurate and mostly conservative constraints to construct an estimated location region that contains the target with high probability. A second

optional phase is to use less accurate and more aggressive constraints to obtain a better estimate of the target location inside the initial estimated region.

In section 4.1.1, we introduced a scheme by which tight bounds can be established for the negative and positive constraints. While that approach, based on computing the convex hull that includes all inter-landmark measurements, achieves high accuracy in practice, it may sometimes return estimated location regions that are too big and imprecise. The reader will observe that it may be possible to use even more aggressive constraints than those dictated by the discussion so far and achieve smaller estimated location regions. The downside of more aggressive constraints is that they may yield an infeasible system of constraints, where the estimated region collapses down to the empty set. In between these two extremes is a setting at which constraints are set just so that the feasible solution space is below a desired parameter. Iterative refinement is an extension to the basic Octant framework to find this setting.

During the calibration phase, Octant additionally computes for each landmark the interpolated spline that minimizes the square error to the latency-to-distance data-points of its inter-landmark measurements, as shown with the dashed lines in Figure 4.4. Opportunistic positive constraints are then derived from the spline by multiplying the spline by a constant δ, while negative constraints are computed by dividing the spline by δ. The value of δ is chosen such that the upper and lower bound contains a given percent of the total number of data points.

Octant initially uses the constraints obtained from the convex hull to compute, typically, a relatively large estimated location area. It then uses this area as a clipping region, which enables it to run through subsequent iterations very

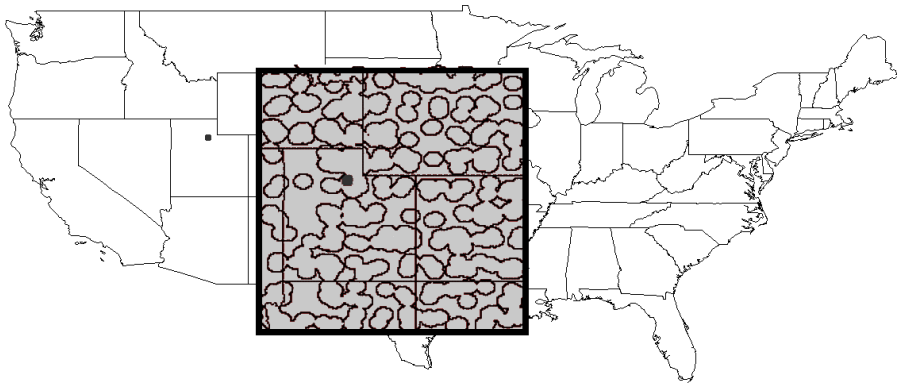


Figure 4.7: Using the city constraints to localize the planetlab2.flux.utah.edu target can significantly reduce the estimated region size as the uninhabited areas between the cities can be removed.

quickly, as it can discard all regions that lie outside the initial solution space. The iterative refinement stage then steps through values for δ, recomputing the estimated location area with successively tighter constraints. The process can terminate when the solution space is below a targeted area, is empty, or when a timeout is reached. This optimization enables Octant to determine how aggressively to extract constraints from the network automatically, without hand tuning.

4.1.6 Geographical Constraints

In addition to constraints extracted from latency measurements, Octant enables any kind of geographical constraint, expressed as arbitrary Bézier-regions, to be integrated into the localization process. In particular, Octant makes it possible to introduce both positive (such as zipcodes from the WHOIS database, zipcodes obtained from other users in the same IP prefix [71]) and negative con-

straints (such as oceans, deserts, uninhabitable areas) stemming from geography and demographics. Clipping estimated location regions with geographic constraints can significantly improve localization accuracy. Since it is highly unlikely for the system to have landmarks in such areas, negative information is typically not available to rule them out. As a result, estimated regions can extend into oceans. In prior work, which does not permit non-convex regions, the removal of such areas typically requires an ad hoc post-processing step. In contrast, Octant can naturally accommodate such constraints during its estimation process. The application of geographic data, such as ocean boundaries, and demographic data, such as population density maps, can vastly improve precision. Figure 4.7 shows the city constraints for a target node in Utah, which is otherwise quite difficult to localize precisely due to its distance to all other landmarks and terrain features.

4.1.7 Point Selection

The Octant approach to localization computes a final estimated localization region which captures the system's best estimate of where the target must lie. Some applications can use such area estimates directly. For instance, web applications might generate content, such as real estate listings, based on the potential zipcodes in which the viewer may be located. Octant can provide the area as either Bézier curve bounded surfaces or an ordered list of coordinates to these applications. Yet many legacy applications, as well as past work, such as GeoPing and GeoTrack, localize targets to a single point. In order to support legacy applications and provide a comparison to previous work, Octant uses a Monte-Carlo algorithm to pick a single point that represents the best estimate

of the target's location. The system initially selects thousands of random points that lie within the estimated region. Each point is assigned a weight based on the sum of its distances to the other selected points. After some number of trials, the point with the least weight is chosen as the best estimate of the target's location. This point is guaranteed to be within the estimated location area by definition, even if the area consists of disjoint regions. Ideally, Octant's point selection interface would only serve in a transitional role for legacy application support. We hope that future applications will be made general enough to take advantage of the extra information in Octant's estimated area.

4.2 Implementation

The Octant framework for geolocalization is practical, entails low measurement overhead and is computationally inexpensive to execute. The core operations involve the manipulation of Bézier curves, which is a compact representation of curves specified by four control points. Standard libraries support common operations, such as intersection, subtraction, and union on Bézier curves, and implement them efficiently by manipulating only the control points [32]. In addition, Bézier curves are robust to slight inaccuracies in their control points [6].

Our Octant implementation does not depend on having control of the target node, or the intermediate routers between the landmarks and the target. Ideally, the target should respond to probes consistently and quickly. A target behind a slow last mile link, or a slow target that incurs high interrupt and processing latencies for all responses will have its response latency factored into its height, which will then compensate for the node's slower speed. Targets that are incon-

sistent can pose problems; our current implementation performs 10 ICMP pings and uses the minimum RTT time as the basis for extracted constraints.

Overall, the code consists of 9800 lines of code, whose structure enables it to operate as a third party service, providing geolocalization results given an IP address using only about 50 nodes deployed on the Internet. When a new landmark comes online, it goes through the calibration phase, measures its latencies to other landmarks, and ships its results back to a centralized server. From then on, the landmarks simply perform ping measurements to target IP addresses and report their results to the centralized server. The server performs the localization by combining the constraints. On a 2GHz machine, this operation currently takes a few seconds once the landmarks are calibrated. The system can easily be made decentralized or optimized further, though our focus has been on improving its accuracy rather than its speed, which we find reasonable.

4.3 Evaluation

We evaluated Octant using physical latency data collected from a PlanetLab dataset consisting of 51 PlanetLab [11] landmark nodes in North America, as well as a public traceroute server dataset consisting of 53 traceroute servers maintained by various commercial and academic institutions in North America. The latency data for the PlanetLab dataset and the public traceroute servers dataset were collected on Feb 1, 2006 and Sept 18, 2006, respectively, using 10 time-dispersed ICMP ping probes to measure round-trip times. Kernel timestamps were used in the latency measurements to minimize timing errors due to overloaded PlanetLab nodes. To evaluate the efficacy of using secondary land-

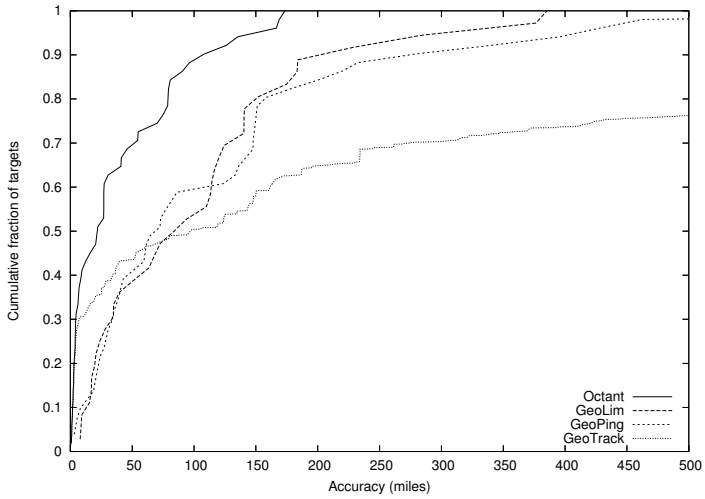


Figure 4.8: Comparison of the accuracy of different localization techniques in the PlanetLab dataset. Octant achieves significantly greater accuracy than previous work, yielding point estimates for nodes that are substantially closer to the real positions of the targets.

marks, we also collected the full traceroute information between every landmark and target pair, as well as latency data between the landmarks and intermediate routers. Whenever appropriate, we repeat measurements 10 times, randomizing landmark selection, and plot the standard deviation.

Evaluating the accuracy of a geolocalization system is difficult, because it necessitates a reliable source of IP to location mappings that can be used as the "ground truth." Such an authoritative mapping is surprisingly difficult to obtain. Our evaluation relies on a total of 104 targets, chosen from the PlanetLab and the traceroute server

datasets as described below. We limit our study to North America as the number of PlanetLab nodes on other continents is too few to yield statistically significant results. We vary the number of landmarks inside North America to

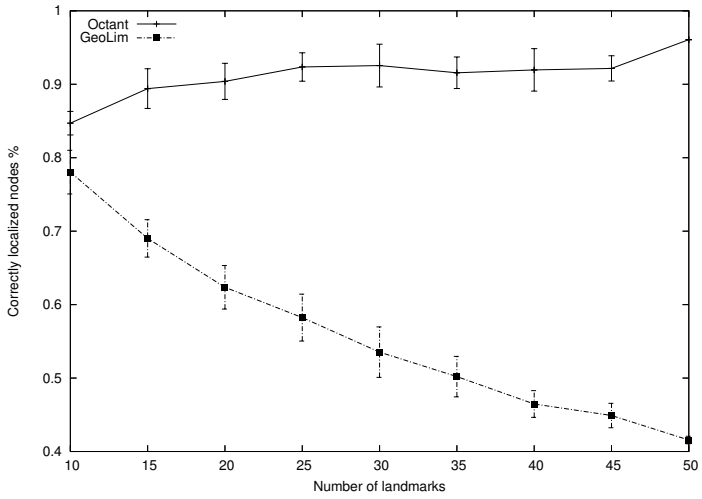


Figure 4.9: The percentage of targets inside the Octant's location estimate is significantly higher than GeoLim's due to Octant's mechanisms for handling uncertainty of individual landmark's location estimate.

examine the behavior of the system at lower densities.

In our PlanetLab dataset, nodes serve both as landmarks and targets, following [71, 38]; of course, the node's own position information is not utilized when it is serving as a target. No two hosts in our evaluation reside in the same institution, which rules out simple yet unrealistic and unscalable solutions to geolocalization that rely on having a nearby landmark for every target.

The traceroute servers in the public traceroute server dataset are used only as targets, with 32 PlanetLab nodes serving as landmarks. The individual traceroute server owners specify the location of their traceroute servers as part of the traceroute service. However, these self-provided locations are often erroneous; we eliminate nodes whose reported positions violate speed of light constraints or disagree with a commercial IP geolocation database [2] from consideration.

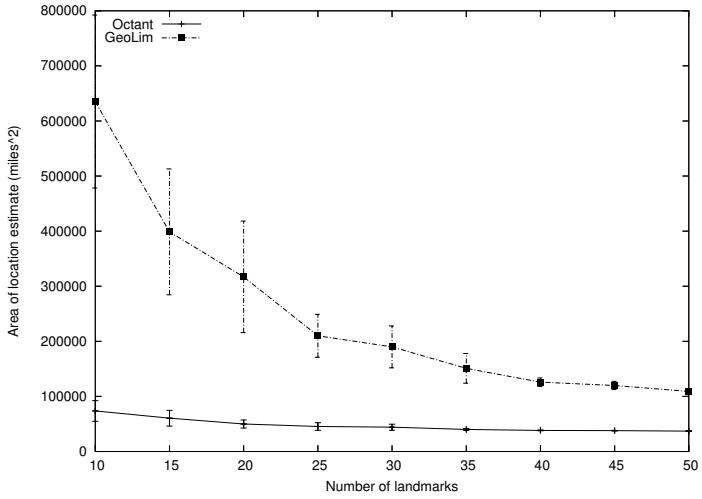


Figure 4.10: The area of Octant's location estimate is significantly smaller than GeoLim's across all tested number of landmarks.

We first compare Octant with GeoLim, GeoPing, and GeoTrack on the PlanetLab dataset. We evaluate these approaches based on their accuracy and precision, and examine how these two metrics vary as a function of the number of landmarks and average inter-landmark distance. We examine accuracy in terms of two metrics: one is the distance between the single point estimate returned by these geolocalization techniques and the physical location of the node, while the other is the percent of nodes whose real-world locations reside within the estimated location area. The former metric allows us to compare Octant to previous systems, such as GeoPing and GeoTrack, that compute only a point estimate, and evaluates how well these systems perform for legacy applications that rely on a single point position. The latter, applicable only to recent geolocalization systems including Octant and GeoLim and proposed by GeoLim [38], evaluates how well area-based approaches perform. Note that comparisons using the latter metric need to be accompanied by measurements on the size of the estimated area (otherwise a simple solution containing the globe will always lo-

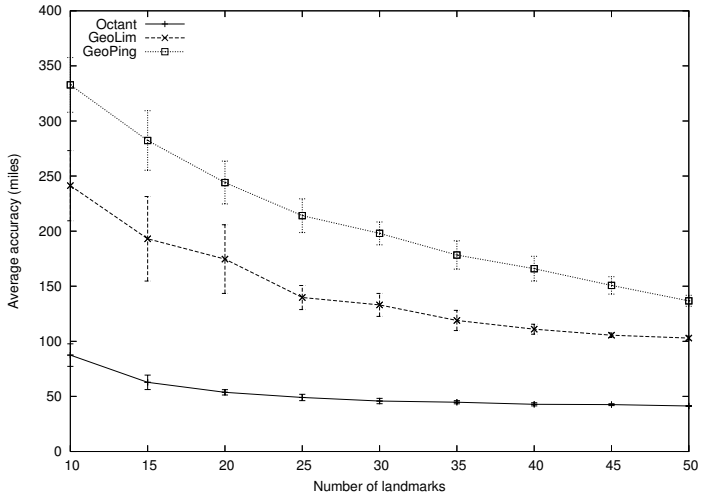


Figure 4.11: The average distance between a target's physical location and the single point estimate returned for that target. Octant achieves high accuracy with low numbers of landmarks.

cate the node accurately), which we also provide.

Figure 4.8 shows the accuracy of different geolocalization techniques by plotting the CDF of the distance between the position estimate and the physical location of a node. Octant significantly outperforms the other three techniques across the entire set of targets. The median accuracy of Octant is 22 miles, compared to median accuracy of 89 miles for GeoLim, 68 miles for GeoPing and 97 miles for GeoTrack. GeoLim was not able to localize approximately 30% of the targets, as its overaggressive constraint extraction leads to empty regions. Even the worst case under Octant is significantly lower than the worst cases encountered with other techniques. The worst case under Octant, GeoLim, GeoPing and GeoTrack are 173, 385, 1071, and 2709 miles, respectively.

A median error of 22 miles is difficult to achieve based solely on constraints obtained online from uncoordinated and unsynchronized hosts, in the presence

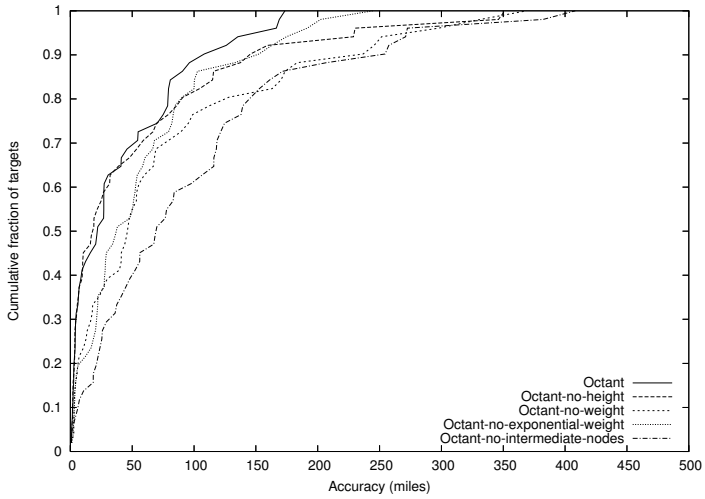


Figure 4.12: The contributions of individual optimizations used in Octant to geolocalization accuracy. The use of intermediate nodes provides the largest improvement to system accuracy.

of routing anomalies and non-great circle routes. As a point of comparison, if all hosts on the Internet were connected via point-to-point fiber links that followed great-circle paths, host clocks were synchronized, and there were no routing anomalies, achieving such error bounds using packet timings would require timing accuracy that could accurately distinguish a delay of $22 * 1.6/(2/3 * 300000) = 170$ microseconds.

In a typical deployment, the number of landmarks used to localize a target is often constrained by physical availability, and an implementation may not be able to use all landmarks in the localization of all targets due to load limits, node failures, or other network management constraints. We evaluate Octant's performance as a function of the number of landmarks used to localize targets, and compare to GeoLim, the only other region-based geolocalization system. Figure 4.9 shows the number of nodes that were located successfully; that is, their physical locations were inside the estimated location region returned by Octant.

Three findings emerge from the plot. First, the percentage of nodes that are successfully localized is quite high for Octant, averaging more than 90% when the number of landmarks exceeds 15. Second, the accuracy of the Octant approach remains flat or improves slightly with increasing numbers of landmarks. Using 15 landmarks yields results that are almost as good as using all 50, and adding more landmarks does not hurt performance. Finally, the accuracy of the GeoLim approach is high for low numbers of landmarks, and drops as more landmarks are introduced. This surprising behavior is due to overaggressive extraction of constraints in GeoLim; as the number of landmarks increases, the chances that a "bad" node, whose network connection yields an unexpected delay, will introduce an over-constraint grows. When there are too many conflicting constraints, GeoLim yields the empty set as its location estimate, whereas the weighted combination of constraints enables Octant to avoid these pitfalls. With all 50 landmarks, GeoLim returns the empty set for more than 29% of the targets.

The preceding evaluation, which examined the percentage of nodes whose physical locations lie inside their estimated location region, needs to be coupled with an examination of the size of the estimated location regions to put the accuracy of the systems in context. Figure 4.10 shows the area of the estimated location region for GeoLim and Octant. The size of the geolocalization region is quite small for Octant at 10 landmarks, and remains the same or slowly decreases with additional landmarks. For small numbers of landmarks, GeoLim returns substantially larger areas that are a factor of 6 bigger than Octant's, which explains its ability to localize about 80% of the nodes with 10 landmarks. Adding more landmarks refines GeoLim's location estimates, though even at 50 landmarks, GeoLim's location estimates are twice the size of Octant's. Octant is

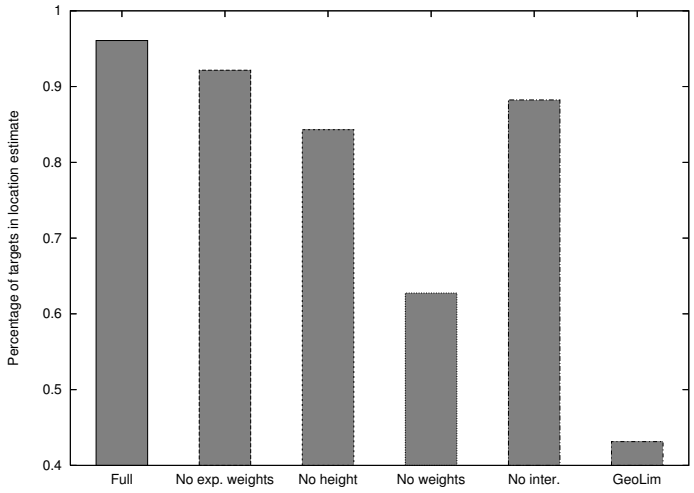


Figure 4.13: The percentage of targets located on average within their estimated location areas for Octant, Octant without various optimizations, and GeoLim.

able to keep the region small via its careful extraction of constraints and use of negative information.

We next examine the absolute error in legacy position estimates based on a single point. Figure 4.11 plots the average distance between a target's physical location and the single point estimate returned for that target. Octant consistently achieves high accuracy, even with landmarks as few as 15. In contrast, GeoLim and GeoPing exhibit performance that degrades as the number of landmarks decreases and their distribution throughout the space becomes more sparse. Octant's performance as the number of landmarks decreases mostly stems from its ability to use routers inside the network as secondary landmarks. Octant's average error is significantly less than both GeoPing and GeoLim even when Octant uses a fifth of the landmarks as the other schemes.

To provide insight into Octant's accuracy, we examine its performance as we

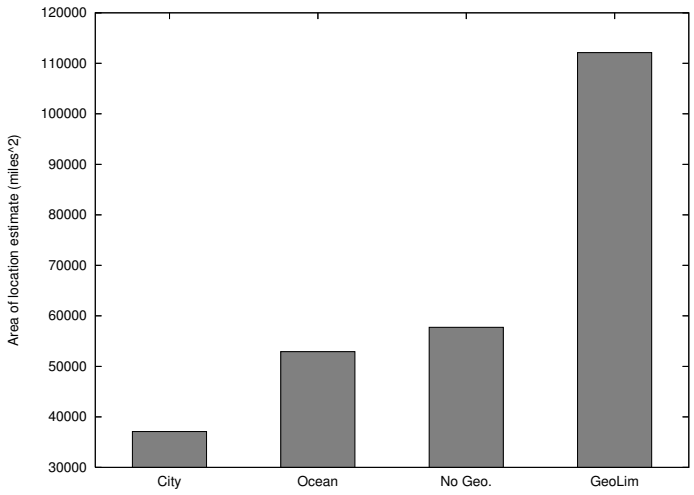


Figure 4.14: The area of the location estimate for Octant with demographic and geographic constraints. The use of these exogenous constraints substantially reduce the size of the estimated area.

disable various optimizations. We examine the individual contribution of each of our optimizations, namely heights, weights, exponential weights and intermediate nodes, by turning off each one in turn and comparing their accuracy with that of the complete Octant system. Figure 4.12 shows the resulting CDFs. The largest improvement to system accuracy is due to the use of intermediate nodes, which significantly increases the number of usable landmarks in the system. **Undns** is useful approximately half the time, but transitive localization is critical to precise localization.

Figures 4.13 and 4.14 provide further insight into the impact of various optimizations on Octant's accuracy and precision. Figure 4.13 plots the percentage of nodes localized with each of the optimizations turned off. NoInter refers to Octant with localization through secondary landmarks inside the network turned off, NoWeights uses no weights associated with constraints, NoHeight disables the last hop delay approximation, NoExpWeights uses weights but all

constraints carry equal weights. The distinction between NoWeights and NoExpWeights is subtle but important. In NoWeights, the estimated location of the target is the intersection of all available constraints. In contrast, NoExpWeights estimates the location of the target as the union of all regions above a certain weight threshold. The effects of a limited number of incorrect constraints can be mitigated by trading off precision, as chosen by the threshold value. The largest contribution in improving accuracy, approximately 33%, stems from the use of weights. GeoLim is less accurate than all the different Octant configurations, even though its location estimates are significantly larger.

The use of geographical constraints in Octant significantly reduces the size of the location estimates. Figure 4.14 shows the size of the location estimates in square miles for Octant with the population density ("cities") constraint which introduces clipping areas into location estimates weighted by the density of the population inside that region, with the oceans constraint which clips oceans from the solution space, and without any geographic constraints, as well as the location estimate area for GeoLim. The use of either geographical constraint alone makes a significant improvement in the location estimate, improving the precision of the estimates. Combined, these geographical estimates greatly improve the fidelity of the location estimates returned by Octant.

The results from our public traceroute servers dataset which includes a mixture of smaller commercial organizations and academic institutions are very similar to those from our PlanetLab dataset. Figure 4.15 shows the CDF of the distance between the position estimate and the physical location of a node. Octant again outperforms the other three techniques across the entire set of targets, with a median localization error of 25 miles, compared to 56 miles for GeoLim,

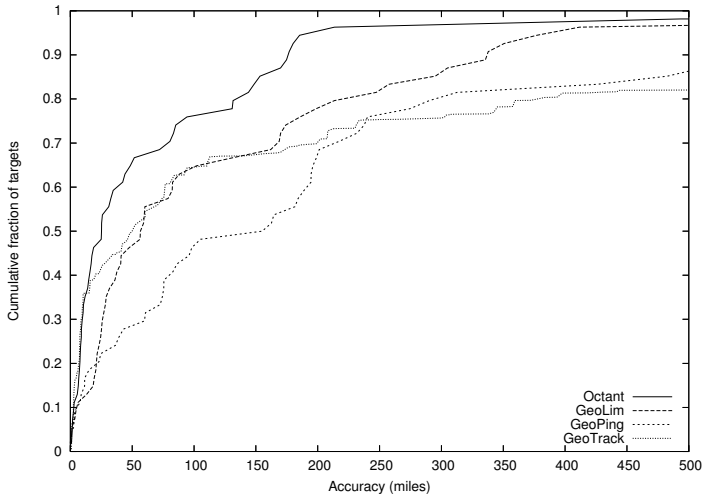


Figure 4.15: The accuracy of different localization techniques on the public traceroute servers dataset show very similar results to those from the PlanetLab dataset, with Octant yielding point estimates that are significant closer to the real positions of the targets.

155 miles for GeoPing and 50 miles for GeoTrack. The significant decrease in accuracy for GeoPing is likely due to the reduction of landmarks from 51 in the PlanetLab dataset to 32 in the traceroute servers dataset, as GeoPing is the most sensitive technique to the number of available landmarks.

These results show that Octant achieves high accuracy in its point estimate for a target, high probability that its location estimate will contain the target, and high precision as indicated by the size of its location estimate area. Overall, Octant can locate the median node to a point within 22 miles of its physical position, or to a tight area (factor of two smaller than previous work) that contains the physical location with 90% probability. Octant requires very few landmarks to be effective; as few as 20 landmarks can achieve approximately the same accuracy as from using all 50 landmarks.

4.4 Summary

Determining the geographic location of Internet hosts is a useful basic building block. Since nodes are identified by IP addresses that are assigned to facilitate efficient routing rather than expose the physical location of nodes, we must instead rely on techniques that can extract location information from network measurements. Octant is a general, comprehensive framework for representing network locations for nodes, extracting constraints on where nodes may or may not be located, and combining these constraints to compute small location estimates that bound the possible location of target nodes with high probability. Octant represents node position and regions precisely using Bézier-bounded regions that can admit any kind of constraint, makes use of both positive and negative information to aggressively reduce the estimated region size, and can effectively reason in the presence of uncertainty and erroneous constraints. It utilizes a number of techniques to extract fine-grain information from end-to-end latency measurements on the Internet. Octant enables network operators to determine, with high confidence, the position of nodes given latency measurements, which in turn enables new location-aware services.

CHAPTER 5

RELATED WORK

The node and object discovery problems addressed in this thesis, namely, network location-aware node selection, decentralized approximate keyword matching, and geolocalization of Internet hosts, are common to many real-world applications and services. As a result, there has been significant past efforts at addressing these problems. In this chapter, I describe these efforts and briefly summarize how they differ from the systems described in this thesis.

5.1 Network Location-Aware Node Selection

Past work on network location services can be separated into approaches that rely on network embeddings and those that do not. I survey both in turn in the next sections.

5.1.1 Network Embedding

Recent work on network coordinates can be categorized roughly into landmark-based systems and simulation-based systems. Both types can embed nodes into a Euclidean coordinate space. Such an embedding allows the distance between any two nodes to be determined without direct measurement.

GNP [69] determines the coordinates of a node by measuring its latency to a fixed set of landmarks and then solving a multidimensional, iterative, non-linear minimization problem. ICS [56] and Virtual Landmarks [93] both aim to reduce the computational cost of the GNP embedding algorithm by replacing it

with a computationally cheaper, linear approximation based on principal component analysis, though the speedup may incur a loss in accuracy. IDES [61] and Phoenix[22] compute coordinates by performing matrix factorization, such as singular value decomposition, of the latency matrix in place of performing a Euclidean embedding. This method can represent non-symmetric routes and latencies that violate triangle inequality but nevertheless achieves similar prediction accuracies as embedding-based systems. Sequoia [74] embed both latency and bandwidth as trees rather than an Euclidean space, and achieves similar latency prediction accuracy as Euclidean embedding-based systems.

To avoid the load imbalance and lack of failure resilience created by using a set of fixed landmarks, PIC [24] and PCoord [55] use landmarks only for bootstrapping and calculate their coordinates based on the coordinates of peers. This can lead to compounding of embedding errors over time in a system with churn. NPS [70] is similar to PIC and PCoord but further imposes a hierarchy on nodes to avoid cyclic dependencies in computing coordinates and to ensure convergence. Lighthouse [73] avoids fixed landmarks entirely and uses multiple local coordinate systems that are joined together through a transition matrix to form a global coordinate system.

Simulation-based systems map nodes and latencies into a physical system whose minimum energy state determines the node coordinates. Vivaldi [25] is based on a simulation of springs, and can be augmented with an additional height vector to increase accuracy. Big-Bang Simulation [86] performs a simulation of a particle explosion under a force field to determine node positions.

IDMaps [34] is a system that can compute the approximate distance between two IP addresses without direct measurement based on strategically placed

tracer nodes. IDMaps incurs inherent errors based on the client's distance to its closest tracer server and requires deploying system wide infrastructure. Other work [33] has also examined how to delegate probing to specialized nodes in the network.

iPlane [57] and iPlane Nano [58] take a structural approach to latency prediction. Extensive traceroutes and latency measurements are combined to construct a partial atlas of the Internet. Segments of measured paths are combined to predict the route between arbitrary hosts. The accuracy of route prediction therefore hinges on having sufficient a priori measurements and these systems can have significant prediction errors in regions that are underrepresented in their respective atlases.

Beverly et al. proposed a machine learning approach to latency prediction [13] by using SVMs to learn the basis of the Internet address space. This system relies on centralize computation on a significant amount of training data, and generally provides prediction accuracies that are worse than embedding-based systems.

Recent theoretical work [51, 88] has sought to explain the empirical success of network embeddings and IDMaps-style approaches. The theoretical foundations for the Meridian framework are presented in detail in an extended technical report [102].

5.1.2 Server Selection

Meridian's closest node discovery protocol draws its inspiration from the Chord DHT [91], which performs routing in a virtual identifier space by halving the virtual distance to the target at each step. Proximity based neighbor selection [19, 20] populates DHT routing tables with nearby nodes, which decreases lookup latency, but does not directly address location-related queries. The time and space complexity of two techniques are discussed in [43] and [46], but these techniques focus exclusively on finding the nearest neighbor, apply only to Internet latencies modeled by growth-constrained metrics, and have not been evaluated with a large scale Internet data.

In beaconing [52], landmark nodes keep track of their latency to all other nodes in the system. A node finds the closest node by querying all landmarks for nodes that are roughly the same distance away from the landmarks. This approach requires each landmark to retain $O(N)$ state, and can only resolve nearest neighbor queries. Binning [77] operates similarly, using approximate bin numbers instead of direct latency measurements. Mithos [97] provides a gradient descent based search protocol to find proximate neighbors in its overlay construction. It is similar to Meridian as it is iterative and performs active probing but it requires $O(N)$ hops to terminate. It is also more prone to terminate prematurely at a local minimum than Meridian as it does not promote diversity in its neighbor set. Various active-probing based nearest neighbor selection schemes are proposed in [85]. These schemes require $O(N)$ state per node, which limits their scalability, and are non-trivial to adapt to other positioning problems. Tiers [9] reduces the state requirement by forming a proximity-aware tree and performing a top-down search to discover the closest node. Hierarchical sys-

tems suffer inherently from load imbalance as nodes close to the root of the hierarchy service more queries, which limits scalability when the workload increases with system size.

Early work on locating nearby copies of replicated services [41] examined combining traceroutes and hop counts to perform a rough triangulation, and to determine the closest replica at a centralized $O(N)$ server using Hotz's distance metric [44]. Dynamic server selection was found in [16] to be more effective than static server selection due to the variability of route latency over time and the large divergence between hop count and latency. Simulations [17] using a simple dynamic server selection policy, where all replica servers are probed and the server with the lowest average latency is selected, show the positive system wide effects of latency-based server selection. Our closest node discovery application can be used to perform such a selection in large-scale networks.

More recently, following my work on ClosestNode.com [100], OASIS [35] provides an alternative DNS to Meridian gateway implementation that explores techniques that trade off localization accuracy for a reduction in on-demand probing. In contrast to ClosestNode.com where nodes in each domain form independent Meridian overlays, OASIS constructs a single Meridian overlay spanning across every domain. It associates each client with the geographic coordinates of the client's closest node in the domain agnostic overlay using the Meridian closest node discovery protocol and caches this association. OASIS answers domain specific queries by using a global lookup table to find the closest node from the requested domain to the client, as a measure of the great circle distance of their associated coordinates. Clients that query multiple OASIS managed domains within a short time-span can take advantage of the caching

to limit on-demand probing to the initial query. However, accuracy suffers as coordinates, with their associated embedding errors, are re-introduced in the node selection process.

5.2 Decentralized Approximate Keyword Matching

The problem of performing decentralized approximate keyword matching can be separated into two primary components: routing in overlay networks and approximate matching techniques. I survey past work on both components in turn in the next sections.

5.2.1 Routing in overlay networks

Cubit is a loosely structured overlay network that most closely resemble a distributed hash table. It differs from previous DHTs [78, 91, 106, 76, 62, 45] by providing a novel match primitive rather than supporting only precise lookups.

Query routing in Cubit is similar to routing in CAN [76] and SWAM [8]. CAN is a coordinate-based approach in which each node knows its immediate closest neighbor in each of the dimensions and greedily routes to the destination. CAN works best when the embedded node set resembles a grid or a torus; it is not designed to work on highly non-homogeneous point sets such as the (embedded) keyword space. Border cases in dealing with churn makes CAN difficult to implement and deploy in practice. SWAM [8] is similar to CAN but partitions the coordinate space into a Voronoi diagram instead of a regular grid.

While the Cubit framework builds on top of Meridian [101], the two systems differ inherently and significantly in the problems they address, the way they perform routing, and the kinds of optimizations they employ. The Cubit framework is more general in that it supports complex boolean search queries and distance metrics beyond network latencies. Cubit queries are also more complex because they require finding the set of *all* nodes that meet a particular constraint, and because Cubit nodes constitute a key/value database instead of the more constrained node/latency space; necessitating significantly different node join and query routing protocols. Cubit also introduces optimizations not applicable to the Meridian context, such as mechanisms to proactively maintain object replication for improved resiliency in a highly dynamic peer-to-peer environment, and to encapsulate and offload keyword regions to nearby neighbors.

Several peer-to-peer systems, e.g. [91, 60, 59], use overlay routing based on small world networks. These systems use a specific virtual space (e.g. a ring) in which long links are introduced such that a simple greedy routing protocol can find short routes. These systems are inherently limited to precise lookups.

5.2.2 Approximate matching

Prior to Cubit, I proposed an alternate design [103] for performing approximate matching that involves representing keywords as points in an Euclidean space. Once nodes and keywords are embedded, techniques such as CAN [76] and Meridian [101] can be used for navigation in that space. While this approach gives a clean and intuitively appealing representation of the keyword space, the literature on metric embeddings does *not* provide embeddings of edit distances

into Euclidean space that are known to have a sufficiently high precision for the approximate matching of short keywords. The embedding is prohibitively inaccurate in practice, distorting the navigation.

Past work has proposed to use the Soundex algorithm to encode keywords by their phonemes before indexing them in a DHT [105]. Unlike edit distance, Soundex is appropriate only for English keywords and is not effective against typing errors.

In DPMS [5], document keywords and search queries are broken up into fixed size substrings. A query match is found if its substrings are a subset of the document's substrings. This matching primitive only accommodates substring matches and does not find near-matches for queries that are misspelled.

Squid [83] creates a multi-dimensional space using a fixed number of keywords as axes. Each object is represented by a set of keywords, and its position in the multi-dimensional space is based on the prefix match distance between the keywords and the axes. The multi-dimensional space is flattened using space filling curves into a one dimensional space, allowing storage and search to be performed on a DHT. This scheme is primarily targeted at range queries on search terms that are small variations of the axes keywords, rather than for arbitrary search terms.

A number of systems make use of coding techniques to provide approximate search. In P2P-AS [65], an error correcting code is introduced that maps small variations of a keyword into the same hash bin. However, the cost of scaling the number of correctable errors is prohibitive. Search terms with more than 3-bits of error require performing additional searches using misspelled variations of

the search terms. Another coding based system is LSH Forest [12], which uses locality-sensitive hashing [37] to cluster similar terms. The system is primarily focused on finding similar documents rather than keywords.

pSearch [92] uses latent semantic indexing on documents to generate vectors that represent its relative similarity to other documents in the system. CAN [76] is used to traverse this vector space. The focus of pSearch is on finding documents with high semantic relevance to the search keys. It is however unable to match misspelled search keys to documents with correctly spelled keywords, as the search keys and keywords may be typographically similar but are semantically unrelated. The computational overhead in using latent semantic indexing is significantly more than edit-distance computations, and the high dimensionality vector spaces created by latent semantic indexing requires a large amount of state to be maintained per CAN node.

5.3 Geolocalization of Internet Hosts

Past work on geolocalization can be broken down into approaches that determine a single point estimate for a target, and those that, like Octant, provide a region encompassing the set of points where the target may lie. With increasing reliance on geolocalization in commerical systems, there have also been significant efforts in identifying the security properties of these approaches.

5.3.1 Single-Point Localization

IP2Geo [71] proposes three different techniques for geolocalization, called GeoPing, GeoTrack and GeoCluster. GeoPing maps the target node to the landmark node that exhibits the closest latency characteristics, based on a metric for similarity of network signatures [7]. The granularity of GeoPing's geolocalization depends on the number and location of the landmarks, requiring a landmark to be close to each target to produce low-error geolocation.

GeoTrack performs a traceroute to a given target, extracts geographical information from the DNS names of routers on the path, and localizes the node to the last router on the path whose position is known. The accuracy of GeoTrack is thus highly dependent on the distance between last recognizable router to the landmark, as well as the accuracy of the positions extracted from router names.

GeoCluster is a database based technique that first breaks the IP address space into clusters that are likely to be geographically co-located, and then assigns a geographical location to each cluster based on IP-to-location mappings from third party databases. These databases include the user registration records from a large web-based e-mail service, a business web-hosting company, as well as the zip-codes of users of an online TV program guide. This technique requires a large, fine-grain and fresh database. Such databases are not readily available to the public due to potential privacy concerns, the clustering may not sufficiently capture locality, the accuracy of such databases must be perpetually refreshed, and, most importantly, the overall scheme is at the mercy of the geographic clustering performed by ISPs when assigning IP address ranges.

Wang et al. [98] proposed a geolocation system that utilizes nodes from

popular online map services that self-report their locations as additional landmarks. Targets are geolocalized to the location of their nearest nodes in the map databases. Much like GeoCluster, this approach relies on having access to a large source of potentially erroneous ground truth data. The accuracy of the system largely depends on the size of the pool of nodes in the map databases that are proximate to the target.

Youn et al. [104] proposed a statistical approach that applies kernel density estimation to delay measurements. The estimated node location is the maximum likelihood position. This statistical approach can be incorporated into the Octant framework and provide a potentially tighter latency-to-distance mapping than the convex hull approach.

Services such as NetGeo [64] and IP2LL [1] geolocalize an IP address using the locations recorded in the WHOIS database for the corresponding IP address block. The granularity of such a scheme is very coarse for large IP address blocks that may contain geographically diverse nodes. The information in the WHOIS database is also not closely regulated and the address information often indicates the location of the head office of the owner which need not be geographically close to the actual target. Quova [3] is a commercial service that provides IP geolocalization based on its own proprietary technique. Neither the details of the technique nor a sample dataset are publicly available.

There are several graphical traceroute tools that offer the geographical location of each intermediate router. GTrace [72] successively uses DNS LOC entries, a proprietary database of domain name to geographical location mappings, NetGeo, and domain name country codes, as available, to localize a given node. VisualRoute [4] is a commercial traceroute tools that also offer geographic

localization of the nodes along the path.

5.3.2 Region Localization

GeoLim [38] derives the estimated position of a node by measuring the network latency to the target from a set of landmarks, extracts upper bounds on position based on inter-landmark distance to latency ratios, and locates the node in the region formed by the intersection of these fixes to established landmarks. Since it does not use negative information, permit non-convex regions or handle uncertainty, this approach breaks down as inter-landmark distances increase.

In contrast, Octant provides a general framework for combining both positive and negative constraints to yield a small, bounded region in which a node is located. It differs from past work in that it enables negative information to be used for localization, separates the selection of a representative point estimate from the computation of the feasible set of points in which a node might be located, permits non-convex solution areas, and aggressively harvests constraints from network latency measurements.

Topology-based Geolocation (TBG) [47] uses the maximum transmission speed of packets in fiber to conservatively determine the convex region where the target lies from network latencies between the landmarks and the target. It additionally uses inter-router latencies on the landmarks to target network paths to find a physical placement of the routers and target that minimizes inconsistencies with the network latencies. TBG relies on a global optimization that minimizes average position error for the routers and target. This process can introduce error in the target position in an effort to reduce errors on the

location of the intermediate routers. Octant differs from TBG by providing a geometric solution technique rather than one based on global optimization. This enables Octant to perform geolocalization in near real-time, where TBG requires significantly more computational time and resources. A geometric solution technique also allows Octant to seamlessly incorporate exogenous geometric constraints stemming from, for example, geography and demographics. This provides Octant with more sources of information for its geolocalization compared to TBG.

A machine-learning based approach, proposed by Eriksson et al. [31], employs a naive Bayesian framework to classify nodes to geographic regions based on prior measurements to these nodes. This approach can robustly combine different measurement sources, but at the cost of reduced fidelity in the geolocalization result. It exhibits lower accuracy than competing approaches using similar sources of measurement data.

Localization has been studied extensively in wireless systems. The wireless localization problem, however, is significantly different from, and easier than, localization on the Internet, as air is close to a perfect medium with well-understood transmission characteristics. The most comprehensive work on localization in wireless networks is Sextant [39]. We share with Sextant the basic insight for accommodating both positive and negative constraints and enabling constraints to be used by landmarks whose positions are not known definitively. Octant differs substantially from Sextant in the various mechanisms it uses to translate Internet measurements to constraints, including its mapping of latencies to constraints, isolating last hop delays, and compensating for indirect routes, among others.

5.3.3 Geolocalization Security

Most geolocalization techniques, including Octant, rely on extracting geographic information from data-sources that can not be entirely trusted. Network measurements to nodes can be altered, and network topological information can be falsified given a sufficiently resourceful attacker.

Muir et al. [66] presents several techniques for adversaries to evade geolocalization systems that rely primarily on registration databases. These techniques include hiding behind proxy servers and altering HTTP headers on requests and responses.

Gill et al. [36] investigates the effectiveness of falsifying network measurements to mislead measurement-based geolocalization systems. The work presents the tradeoffs between the severity and the detectability of different attacks from adversaries of varying resourcefulness. The adversaries span end-users who are limited to increasing network latencies to corporations that can present altered views of the network to the geolocalization systems.

CHAPTER 6

FUTURE DIRECTIONS AND SUMMARY

A number of research directions follow from the work described in this thesis. They span using the space-based approach to address problems in emerging areas, such as cloud-computing, to extending the problems addressed in this thesis to encompass much broader requirements, such as geolocalization at a planet-wide scale. The following sections describe some exemplary directions where the approach advocated in this thesis can potentially yield significant improvements over past work.

Searchable Cloud-based Storage

The emergence of cloud-computing changes a number of assumptions regarding the requirements of operating a large-scale distributed system. A significant body of work has focused on providing scalable, large-scale, cloud-based storage. Current deployed cloud-storage systems, such as BigTable [21] and Cassandra [53], offer a simple key-value storage interface that trades off reduced query complexity for improved scalability and performance. Applications that use these systems must currently work around this limited interface, often resulting in non-intuitive and inconsistent organization of the application data.

The space-based approach can be applied to design a distributed storage system that can support complex search queries while offering similar performance to key-value storage systems. Rather than representing structured data, containing values for multiple attributes, as rows in flat, two-dimensional tables, data can instead be mapped to a position in a multi-dimensional space

where the data attributes serve as the dimensional axes. Complex search queries that specify values on multiple attributes map to search hyperplanes, representing the search regions. These search hyperplanes significantly prune the search space. With proper layout of nodes in the space, only a limited number of nodes need to be queried for each search query. I have designed and implemented HyperDex based on the above design, which is the first project in a line of research that aims to explore the fundamental tradeoffs between capabilities and performance in large-scale, cloud-storage systems.

Passive and Scalable Geolocalization

The geolocalization problem offers additional challenges beyond those tackled by this thesis. In certain contexts, it may be undesirable to perform active measurements on the geolocalization target, as targets that are aware of geolocalization attempts can try to mask their locations or vary the latency of measurement probes to mislead geolocalization systems [36]. In other contexts, there may be a large number of targets to geolocalize at once, requiring a scalable geolocalization approach.

The Octant framework described in this thesis provides a principled foundation for meeting these challenges. It is straightforward to adopt the Octant framework to use passive measurements in lieu of active measurement probes. Passive measurements eliminate the measurement delays introduced by active measurement probes, and provide the additional benefit of ensuring the targets are oblivious of the geolocalization attempts.

For contexts that require extreme scalability, the Octant framework can be re-

structured into a multi-stage Map/Reduce process that can distribute the highly parallelizable computational workload across the nodes in a datacenter. Further scalability improvements can come from aggregating latency measurements into a large topological map that significantly reduces the input size without any loss of accuracy or precision.

Summary of Thesis

Node and object discovery problems are pervasive in large-scale distributed systems, affecting performance, scalability and correctness if they are not adequately addressed. Past work introduced many different ad-hoc techniques for addressing each specialized discovery problem. In this thesis, I introduced a unifying, space-based approach that is effective at addressing a diverse set of node and discovery problems and significantly improves upon the performance and accuracy of past solutions.

The crux of the space-based approach is to model and solve distributed problems geometrically. The challenge in applying this approach lies in finding an appropriate space to represent the problem and designing efficient distributed data-structures and algorithms for traversing this space. The solution regions, representing the collection of locations that represent solutions to the problem, are often immediately apparent once appropriate spaces are found. An important design decision in my systems is to create spaces that utilize the problems' measure of success, namely network latency, keyword edit-distance, and geographic distance, as the distance metric. This design, in contrast to designs based on embedding, forgo introducing additional structures that can aid in

traversal of the spaces but create distortions or artifacts that can affect the accuracy of the solutions. For the problems I explored in this thesis, I demonstrated that the structure introduced by embedding systems is neither necessary nor desirable; the respective solution spaces can be efficiently traversed directly, and avoiding an embedding yields substantial improvements in accuracy.

I tackled three node and object discovery problems in this thesis to illustrate the flexibility of the space-based approach. The problems are latency-aware node selection, decentralized approximate keyword matching, and geolocalization of Internet hosts. These problems are the key challenges to providing efficient content-distribution, fully decentralized file-sharing networks, and context-aware advertisement delivery. Using the space-based approach to solve these problems paid significant dividends; it provided a framework for computing solutions precisely and efficiently, and enabled techniques and optimizations to be shared across the different solutions. The three systems I built for solving these problems provided significant improvements to both performance and accuracy over past work. Of equal importance, these systems have also served as vehicles for many other research projects, both past and on-going, and I hope they will continue to play a role in shaping future distributed system designs.

APPENDIX A

K-CLOSEST NODE DISCOVERY IN MQL

```
1  Measurement[] sort_measure(int k, Measurement r_lat[]) {
2    double d_lat[];
3    for (int i = 0; i < array_size(r_lat); i = i + 1) {
4      push_back(d_lat, r_lat[i].distance[0]);
5    }
6    Measurement ret_list[];
7    while (array_size(ret_list) < k && array_size(d_lat) > 0) {
8      int offset = array_min_offset(d_lat);
9      push_back(ret_list, r_lat[offset]);
10     d_lat[offset] = d_lat[array_size(d_lat)-1];
11     r_lat[offset] = r_lat[array_size(r_lat)-1];
12     pop_back(d_lat);
13     pop_back(r_lat);
14   }
15   return ret_list;
16 }
17
18 Measurement[] local_closest(int k, double beta, Node t) {
19   int ping_TO = 1000;
20   Node ts[] = {t};
21   Measurement self = get_distance_icmp(ts, ping_TO);
22   double self_lat = self.distance[0];
23   if (self_lat > dbl(ping_TO)) {
24     Measurement empty[];
25     return empty;
26   }
27   Node ring_m[] = array_intersect(
28       ring_ge((1.0 - beta) * self_lat), ring_le((1.0 + beta) * self_lat));
29   if (array_size(ring_m) == 0) {
30     Measurement ret_list[]= {self};
31     return ret_list;
32   }
33   int timeout = ceil(2.0 * (1.0 + beta) * self_lat);
34   Measurement r_lat[] = get_distance_icmp(ring_m, ts, timeout);
35   push_back(r_lat, self);
36   r_lat = sort_measure(k, r_lat);
37   Measurement r_ret[];
38   for (int i = 0; i < array_size(r_lat); i = i + 1) {
39     if (r_lat[i].distance[0] > dbl(timeout)) {
40       break;
41     }
42     push_back(r_ret, r_lat[i]);
43   }
44   return r_ret;
45 }
46
47 int is_in_list(Node c_list[], Node n_node) {
48   for (int i = 0; i < array_size(c_list); i = i + 1) {
49     if (c_list[i].addr == n_node.addr &&
50         c_list[i].port == n_node.port) {
51       return 1;
```

```
52      }
53    }
54    return 0;
55  }
56
57  Node[] filter_list(Node n_list[], Measurement f_list[]) {
58    Node r_list[];
59    for (int i = 0; i < array_size(n_list); i = i + 1) {
60      Node c_node = n_list[i];
61      for (int j = 0; j < array_size(f_list); j = j + 1) {
62        Measurement m = f_list[j];
63        if (c_node.addr == m.addr && c_node.port == m.port) {
64          push_back(r_list, c_node);
65          break;
66        }
67      }
68    }
69    return r_list;
70  }
71
72  Node[] measure_to_list(Measurement f_list[]) {
73    Node r_list[];
74    for (int i = 0; i < array_size(f_list); i = i + 1) {
75      Measurement r = f_list[i];
76      Node n = {r.addr, r.port, 0, 0};
77      push_back(r_list, n);
78    }
79    return r_list;
80  }
81
82  Measurement[] get_closest(Node t, int k) {
83    Node c_list[];
84    Measurement f_list[] = local_closest(k, 0.5, t);
85    Measurement s_list[];
86    for (int i = 0; i < array_size(f_list); i = i + 1) {
87      if (f_list[i].addr == 0) {  // Check if it is the local node
88        push_back(s_list, f_list[i]);
89        f_list[i] = f_list[array_size(f_list)-1];
90        pop_back(f_list);
91        f_list = sort_measure(k, f_list);
92        break;
93      }
94    }
95    Node n_list[] = measure_to_list(f_list);
96    while (array_size(n_list) > 0) {
97      Node e = n_list[array_size(n_list)-1];
98      pop_back(n_list);
99      Measurement r_list[] = rpc(e, local_closest, k, 0.5, t);
100     push_back(c_list, e);
101     for (int i = 0; i < array_size(r_list); i = i + 1) {
102       Measurement r = r_list[i];
103       if (r.addr == 0) {
104         r.addr = e.addr; r.port = e.port;
105       }
106       Node n_node = {r.addr, r.port, 0, 0};
```

```
107       if (is_in_list(c_list, n_node) == 1) {
108         continue;
109       }
110       if (is_in_list(n_list, n_node) == 0) {
111         f_list = sort_measure(k, f_list);
112         if (array_size(f_list) < k) {
113           push_back(f_list, r);
114           push_back(n_list, n_node);
115         } else if (r.distance[0] < f_list[k-1].distance[0]) {
116           push_back(f_list, r);
117           push_back(n_list, n_node);
118         }
119       }
120     }
121     f_list = sort_measure(k, f_list);
122     n_list = filter_list(n_list, f_list);
123   }
124   if (array_size(f_list) == 0) {
125     return s_list;
126   }
127   return f_list;
128 }
```

BIBLIOGRAPHY

[1] IP to Latitude/Longitude Server, University of Illinois. http://cello.cs.uiuc.edu/cgi-bin/slamm/ip2ll.

[2] IP2Location. http://www.ip2location.com.

[3] Quova. http://www.quova.com.

[4] Visualware Inc. http://www.visualroute.com.

[5] R. Ahmed and R. Boutaba. Distributed pattern matching: A key to flexible and efficient p2p search. *IEEE Journal on Selected Areas in Communications*, 25(1), 2007.

[6] D. Assaf. *The Sensitivity of Spline Functions on Triangulations to Vertex Perturbation*. PhD thesis, Vanderbilt University, May 1998.

[7] P. Bahl and V. Padmanabhan. RADAR: An In-Building RF-Based User Location and Tracking System. In *INFOCOM*, Tel Aviv, Israel, March 2000.

[8] F. Banaei-Kashani and C. Shahabi. Swam: A family of access methods for similarity-search in peer-to-peer data networks. In *CIKM*, 2004.

[9] S. Banerjee, C. Kommareddy, and B. Bhattacharjee. Scalable Peer Finding on the Internet. In *Global Internet Symposium*, Taipei, Taiwan, November 2002.

[10] C. Barber, D. Dobkin, and H. Huhdanpaa. The Quickhull Algorithm for Convex Hulls. *Transactions on Mathematical Software*, 22(4), 1996.

[11] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating System Support for Planetary-Scale Network Services. In *Networked Systems Design and Implementation*, San Francisco, CA, March 2004.

[12] M. Bawa, T. Condie, and P. Ganesan. Lsh forest: Self-tuning indexes for similarity search. In *WWW*, Japan, 2005.

[13] R. Beverly, K. Sollins, and A. Berger. SVM Learning of IP Address Structure for Latency Prediction. In *MineNet*, Pisa, Italy, 2006.

[14] BitTorrent.com. Advanced bittorrent, 2010. http://www.bittorrent.com/btusers/guides/bittorrent-user-manual/appendix-bittorrentmainline-interface/-preferences/advanced#dht.rate.

[15] J. Cao, D. Davis, S. Wiel, and B. Yu. Time-varying Network Tomography. *American Statistical Association*, 95, 2000.

[16] R. Carter and M. Crovella. Server Selection Using Dynamic Path Characterization in Wide-Area Networks. In *INFOCOM*, Kobe, Japan, April 1997.

[17] R. Carter and M. Crovella. On the Network Impact of Dynamic Server Selection. *Computer Networks*, 31, 1999.

[18] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *OSDI*, MA, 2002.

[19] M. Castro, P. Druschel, Y. Hu, and A. Rowstron. Exploiting Network Proximity in Peer-to-Peer Overlay Networks. In *Technical Report MSR-TR-2003-82*, Microsoft Research, 2002.

[20] M. Castro, P. Druschel, Y. Hu, and A. Rowstron. Proximity Neighbor Selection in Tree-Based Structured Peer-to-Peer Overlays. In *Technical Report MSR-TR-2003-52*, Microsoft Research, 2003.

[21] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. In *Operating Systems Design and Implementation*, 2006.

[22] Y. Chen, X. Wang, X. Song, E. Lua, C. Shi, X. Zhao, B. Deng, and X. Li. Phoenix: Towards an Accurate, Practical and Decentralized Network Coordinate System.

[23] CiteSeer.IST. Citeseer publications researchindex, 2008. http://citeseer.ist.psu.edu/.

[24] M. Costa, M. Castro, A. Rowstron, and P. Key. PIC: Practical Internet Coordinates for Distance Estimation. In *Intl. Conference on Distributed Computing Systems*, Tokyo, Japan, March 2004.

[25] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A Decentralized Network Coordinate System. In *SIGCOMM*, Portland, OR, August 2004.

[26] F. Dabek, M. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-Area Cooperative Storage with CFS. In *Symposium on Operating Systems Principles*, Banff, AB, Canada, October 2001.

[27] E. Damiani, D. C. d. Vimercati, S. Paraboschi, P. Samarati, and F. Violante. A reputation-based approach for choosing reliable resources in peer-to-peer networks. In *CCS*, 2002.

[28] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*, San Francisco, CA, December 2004.

[29] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic Algorithms for Replicated Database Maintenance. In *Symposium on Principles of Distributed Computing*, Vancouver, BC, Canada, August 1987.

[30] J. R. Douceur. The sybil attack. In *IPTPS*, MA, 2002.

[31] B. Eriksson, P. Barford, J. Sommers, and R. Nowak. A Learning-based Approach for IP Geolocation. In *PAM*, Zurich, Switzerland, 2010.

[32] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide*. Academic Press, 1988.

[33] Z. Fei, S. Bhattacharjee, E. Zegura, and M. Ammar. A Novel Server Selection Technique for Improving the Response Time of a Replicated Service. In *INFOCOM*, San Francisco, CA, March 1998.

[34] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. IDMaps: A Global Internet Host Distance Estimation Service. *Transactions on Networking*, 9:525–540, October 2001.

[35] M. Freedman, K. Lakshminarayanan, and D. Mazieres. OASIS: Anycast for Any Service. In *NSDI*, San Jose, CA, May 2006.

[36] P. Gill, Y. Ganjali, B. Wong, and D. Lie. Dude where's that IP? Circumventing Measurement-based IP Geolocation. In *USENIX Security*, Washington, DC, August 2010.

[37] A. Gionis, P. Indyk, and R. Motwani. Similarity Search in High Dimensions viah Hashing. In *VLDB*, Edinburgh, Scotland, 1999.

[38] B. Gueye, A. Ziviani, M. Crovella, and S. Fdida. Constraint-Based Geolocation of Internet Hosts. In *Internet Measurement Conference*, Taormina, Sicily, Italy, October 2004.

[39] S. Guha, R. Murty, and E.G. Sirer. Sextant: A Unified Framework for Node and Event Localization in Sensor Networks. In *Mobihoc*, Urbana-Champaign, IL, May 2005.

[40] K. Gummadi, S. Saroiu, and S. Gribble. King: Estimating Latency between Arbitrary Internet End Hosts. In *Internet Measurement Workshop*, Marseille, France, November 2002.

[41] J. Guyton and M. Schwartz. Locating Nearby Copies of Replicated Internet Servers. In *SIGCOMM*, Cambridge, MA, August 1995.

[42] A. Haeberlen, P. Kouznetsov, and P. Druschel. Peerreview: Practical accountability for distributed systems. In *SOSP*, WA, 2007.

[43] K. Hildrum, J. Kubiatowicz, S. Rao, and B. Zhao. Distributed Object Location in a Dynamic Network. In *Symposium on Parallel Algorithms and Architectures*, Winnipeg, MB, Canada, August 2002.

[44] S. Hotz. *Routing Information Organization to Support Scalable Interdomain Routing with Heterogeneous Path Requirements*. PhD thesis, Univ. of Southern California, 1994.

[45] F. Kaashoek and D. Karger. Koorde: A simple degree-optimal distributed hash table. In *IPTPS*, CA, 2003.

[46] D. Karger and M. Ruhl. Finding Nearest Neighbors in Growth-restricted Metrics. In *Symposium on Theory of Computing*, Montreal, QC, Canada, May 2002.

[47] E. Katz-Bassett, J. John, A. Krishnamurthy, D. Wetherall, T. Anderson, and Y. Chawathe. Towards IP Geolocation using Delay and Topology Measurements. In *Internet Measurement Conference*, Rio de Janeiro, Brazil, October 2006.

[48] S. H. Khor, N. Christin, T. Wong, and A. Nakao. Power to the People:

Securing the Internet One Edge at a Time. In *LSAD*, Kyoto, Japan, August 2007.

[49] J. Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. In *STOC*, Portland, OR, May 2000.

[50] J. Kleinberg. Complex networks and decentralized search algorithms. In *Intl. Congress of Mathematicians*, 2006.

[51] J. Kleinberg, A. Slivkins, and T. Wexler. Triangulation and Embedding Using Small Sets of Beacons. *J. of the ACM*, 56(6), September 2009. Preliminary version has appeared in *45th IEEE FOCS*, 2004.

[52] C. Kommareddy, N. Shankar, and B. Bhattacharjee. Finding Close Friends on the Internet. In *Intl. Conference on Network Protocols*, Riverside, CA, November 2001.

[53] A. Lakshman and P. Malik. Cassandra - a decentralized structured storage system. In *SOSP Workshop on Large Scale Distributed Systems and Middleware*, 2009.

[54] J. Ledlie, P. Gardner, and M. Seltzer. Network Coordinates in the Wild. In *NSDI*, Cambridge, MA, April 2007.

[55] L. Lehman and S. Lerman. PCoord: Network Position Estimation Using Peer-to-Peer Measurements. In *Intl. Symposium on Network Computing and Applications*, Cambridge, MA, August 2004.

[56] H. Lim, J. Hou, and C. Choi. Constructing Internet Coordinate System Based on Delay Measurement. In *Internet Measurement Conference*, Miami, Florida, October 2003.

[57] H. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iplane: An information plane for distributed services. In *OSDI*, Seattle, WA, 2006.

[58] H. Madhyastha, E. Katz-Bassett, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane Nano: Path Prediction for Peer-to-Peer Applications, 2009.

[59] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *PODC*, CA, 2002.

[60] G. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed hashing in a small world. In *USITS*, WA, 2003.

[61] Y. Mao, L. Saul, and J. Smith. IDES: An Internet Distance Estimation Service for Large Networks. 24, 2006.

[62] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS*, MA, 2002.

[63] R. C. Merkle. Secure communications over insecure channels. *Communications of the ACM*, 1978.

[64] D. Moore, R. Periakaruppan, and J. Donohoe. Where in the World is netgeo.caida.org? In *INET2000 Poster*, Yokohama, Japan, July 2000.

[65] A. Mowat, R. Schmidt, M. Schumacher, and I. Constantinescu. Extending peer-to-peer networks for approximate search. In *SAC*, Brazil, 2008.

[66] J. Muir and P. Oorschot. Internet Geolocation: Evasion and Counterevasion. *ACM Computing Survey*, 42, December 2009.

[67] H. Neemuchwala and A. Hero. Image registration in high-dimensional feature space. In *Computational Imaging*, CA, 2005.

[68] Netflix. Netflix prize, 2006. http://www.netflixprize.com.

[69] T. Ng and H. Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *INFOCOM*, New York, NY, June 2002.

[70] T. Ng and H. Zhang. A Network Positioning System for the Internet. In *USENIX*, Boston, MA, June 2004.

[71] V. Padmanabhan and L. Subramanian. An Investigation of Geographic Mapping Techniques for Internet Hosts. In *SIGCOMM*, San Diego, CA, August 2001.

[72] R. Periakaruppan and E. Nemeth. GTrace - A Graphical Traceroute Tool. In *LISA*, Seattle, WA, November 1999.

[73] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti. Lighthouses for Scalable Distributed Location. In *Intl. Workshop on Peer-To-Peer Systems*, Berkeley, CA, February 2003.

[74] V. Ramasubramanian, D. Malkhi, F. Kuhn, M. Balakrishnan, A. Gupta, and A. Akella. On the Treeness of Internet Latency and Bandwidth. In *SIGMETRICS*, Seattle, WA, 2009.

[75] V. Ramasubramanian and E. G. Sirer. Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays. In *NSDI*, CA, 2004.

[76] S. Ratnasamy, P. Francis, M. Hadley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *SIGCOMM*, San Diego, CA, August 2001.

[77] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-Aware Overlay Construction and Server Selection. In *INFOCOM*, New York, NY, June 2002.

[78] A. Rowstron and P. Druschel. Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *Middleware*, Heidelberg, Germany, November 2001.

[79] A. Rowstron and P. Druschel. Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility. In *Symposium on Operating Systems Principles*, Banff, AB, Canada, October 2001.

[80] S. Savage, A. Collins, and E. Hoffman. The End-to-End Effects of Internet Path Selection. In *SIGCOMM*, Cambridge, MA, September 1999.

[81] M. Scheidegger and T. Braun. Improved Locality-Aware Grouping in Overlay Networks. In *KiVS*, Bern, Switzerland, February 2007.

[82] M. Scheidegger and T. Braun. Meridian-based Grouping in Overlay Networks. *it - Information Technology*, 49, 2007.

[83] C. Schmidt and M. Parashar. Flexible information discovery in decentralized distributed systems. In *HPDC*, WA, 2003.

[84] Searchspell. Searchspell typo, 2010. http://www.searchspell.com/typo/.

[85] K. Shanahan and M. Freedman. Locality Prediction for Oblivious Clients. In *Intl. Workshop on Peer-To-Peer Systems*, Ithaca, NY, February 2005.

[86] Y. Shavitt and T. Tankel. Big-Bang Simulation for Embedding Network Distances in Euclidean Space. In *INFOCOM*, San Francisco, CA, April 2003.

[87] A. Slivkins. Distance estimation and object location via rings of neighbors. In *PODC*, Las Vegas, NV, 2005.

[88] A. Slivkins. Distributed Approaches to Triangulation and Embedding. In *the Symposium on Discrete Algorithms*, Vancouver, BC, Canada, January 2005.

[89] Y.J. Song, V. Ramasubramanian, and E.G. Sirer. Optimal Resource Utilization in Content Distribution Networks. In *Computing and Information Science Technical Report TR2005-2004*, Cornell University, November 2005.

[90] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP Topologies with Rocketfuel. In *SIGCOMM*, Pittsburgh, PA, August 2002.

[91] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *SIGCOMM*, San Diego, CA, August 2001.

[92] C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *SIGCOMM*, Germany, 2003.

[93] L. Tang and M. Crovella. Virtual Landmarks for the Internet. In *Internet Measurement Conference*, Miami, Florida, October 2003.

[94] D. Tennenhouse and D. Wetherall. Towards an Active Network Architecture. *Computer Communication Review*, 26(2), 1996.

[95] Adam Twiss, Mike Belshe, and Michael Campanella. Apache HTTP server benchmarking tool. http://httpd.apache.org/docs/2.0/programs/ab.html.

[96] Y. Vardi. Network Tomography: Estimating Source-Destination Traffic Intensities from Link Data. *American Statistical Association*, 91, 1996.

[97] M. Waldvogel and R. Rinaldi. Efficient Topology-Aware Overlay Network. In *Hot Topics in Networks*, Princeton, NJ, October 2002.

[98] Y. Wang, D. Burgener, M. Flores, A. Kuzmanovic, and C. Huang. Towards Street-Level Client-Independent IP Geolocation. In *NSDI*, Boston, MA, 2011.

[99] B. Wong. Cubit: Approximate matching for peer-to-peer overlays, 2011. http://www.cs.cornell.edu/~bwong/cubit.

[100] B. Wong and E.G. Sirer. ClosestNode.com: An Open-Access, Scalable, Shared Geocast Service for Distributed Systems. *SIGOPS Operating Systems Review*, 40(1), 2006.

[101] B. Wong, A. Slivkins, and E. G. Sirer. Meridian: A lightweight network location service without virtual coordinates. In *SIGCOMM*, PA, 2005.

[102] B. Wong, A. Slivkins, and E.G. Sirer. Meridian: A Lightweight Network Location Service without Virtual Coordinates. In *Computing and Information Science Technical Report TR2005-1982*, Cornell University, May 2005.

[103] B. Wong, Y. Vigfússon, and E. G. Sirer. Hyperspaces for object clustering and approximate matching in peer-to-peer overlays. In *HotOS*, 2007.

[104] I. Youn, B. Mark, and D. Richards. Statistical Geolocation of Internet Hosts. In *ICCCN*, San Francisco, CA, 2009.

[105] M.A. Zaharia, A. Chandel, S. Saroiu, and S. Keshav. Finding content in file-sharing networks when you can't even spell. In *Intl. Workshop on P2P Systems*, WA, 2007.

[106] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing. In *Technical Report UCB/CSD-01-1141*, UC Berkeley, April 2001.