Meridian: A Lightweight Network Location Service without Virtual Coordinates

Bernard Wong Aleksandrs Slivkins Emin Gün Sirer
Dept. of Computer Science, Cornell University, Ithaca, NY 14853
{bwong, slivkins, egs}@cs.cornell.edu

ABSTRACT

This paper introduces a lightweight, scalable and accurate framework, called Meridian, for performing node selection based on network location. The framework consists of an overlay network structured around multi-resolution rings, query routing with direct measurements, and gossip protocols for dissemination. We show how this framework can be used to address three commonly encountered problems, namely, closest node discovery, central leader election, and locating nodes that satisfy target latency constraints in large-scale distributed systems without having to compute absolute coordinates. We show analytically that the framework is scalable with logarithmic convergence when Internet latencies are modeled as a growth-constrained metric, a low-dimensional Euclidean metric, or a metric of low doubling dimension. Large scale simulations, based on latency measurements from 6.25 million node-pairs as well as an implementation deployed on PlanetLab show that the framework is accurate and effective.

Categories and Subject Descriptors: C.2.1 **[Computer-Communication Networks]**: Network Architecture and Design—*Network topology*

General Terms: Algorithms, Design, Measurement, Performance

Keywords: Node selection, Network locality, Nearest neighbor

1. INTRODUCTION

Selecting nodes based on their location in the network is a basic building block for many 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.

A general technique for finding nodes that optimize a given network metric is to perform a *network embedding*, that is, to map high-dimensional network measurements into a location in a smaller Euclidean space. For instance, recent work in network positioning [42, 15, 40, 57, 52, 44, 12, 43, 39] uses large vectors of node-to-node latency measurements on the Internet to determine a corresponding single point in a d-dimensional space for each node. The resulting embedded address, a *virtual coordinate*, can be used to select nodes.

While the network embedding approach is applicable for a wide range of applications, it is neither accurate nor complete. The embedding process typically introduces significant errors. Selection of parameters, such as the constant d, the set of measurements taken to perform the embedding, the landmarks used for measurement, and the timing interval in which measurements are taken, is non-trivial and has a significant impact on the accuracy of the approach. Further, coordinates need to be recomputed as network latencies fluctuate. In addition, complex mechanisms besides virtual coordinates are required to support large-scale applications. Simple schemes, such as centralized servers that retain $O(N)$ state or naive algorithms with $O(N)$ running time, are unsuitable for large-scale networks. Peer-to-peer substrates that can naturally work with Euclidean coordinates and support range queries, such as CAN [46], Mercury [5] and P-Trees [13], can reduce the state requirements per node; however, these systems introduce substantial complexity and bandwidth overhead in addition to the overhead of network embedding. And our simulation results show that, even with a P2P substrate that always finds the best node based on virtual coordinates, the embedding error leads to a suboptimal choice.

This paper introduces a lightweight, scalable and accurate framework, called Meridian, for performing node selection based on network location. Meridian forms a loosely-structured overlay network, uses direct measurements instead of a network embedding, and can solve spatial queries without an absolute coordinate space. Its functionality is similar to that of GNP combined with CAN in performing node selection based on network location[1].

Each Meridian node keeps track of $O(\log N)$ peers and organizes them into concentric rings of exponentially increasing radii. A query is matched against the relevant nodes in these rings, and optionally forwarded to a subset of the node's peers. Intuitively, the forwarding "zooms in" towards the solution space, handing off the query to a node that has more information to solve the problem due to the structure of its peer set. A scalable gossip protocol is used to notify other nodes of membership in the system. A node selection algorithm provides diverse ring membership to maximize the marginal utility provided by each ring member. Meridian avoids incurring embedding errors by making no attempt to reconcile the latencies seen at participating nodes into a globally consistent coordinate space. Directly evaluating queries against relevant peers in each ring avoids errors stemming from out of date coordinates. Meridian provides a general framework applicable for a wide range of network location problems. In this paper, we focus on three network location problems that are commonly encountered

SIGCOMM'05, August 21–26, 2005, Philadelphia, Pennsylvania, USA.

[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.

in distributed systems, and describe how the lightweight Meridian framework can be used to address them.

The first network location problem that we examine is that of discovering the closest node to a targeted reference point. This is a pervasive problem; content distribution networks (CDNs) [29], large-scale multiplayer games [38], and peer-to-peer overlays [27, 30, 9, 8], among others, can significantly reduce response time and network load by selecting nodes close to targets. For instance, a geographically distributed peer-to-peer web crawler can reduce crawl time and minimize network load by delegating the crawl to the closest node to each target web server, CDNs can reduce latency by assigning clients to nearby servers, and multiplayer games can improve gameplay by reducing server latency.

Meridian can also be used to find a node that offers minimal latencies to a given set of nodes. Intuitively, various applications seek to locate a node that is at the centerpoint of the region defined by the set members. This basic operation can be used for location-aware leader election, where the chosen central node enables average communication latency to be minimized. For instance, an application-level multicast system can use location-aware leader election to improve transmission latencies by placing centrally-located nodes higher in the tree.

Finally, we examine the problem of finding a set of nodes in a region whose boundaries are defined by latency constraints. For instance, given a set of latency constraints to well-known peering points, we show how Meridian can locate nodes in the region defined by the intersection of these constraints. This functionality is useful for ISPs and hosting services to cost effectively meet service-level agreements, for computational grids to locate nodes with specific inter-cluster latency requirements, and generally, for applications that require fine-grain selection of services based on latency to multiple targets.

We demonstrate through a theoretical analysis that our system provides robust performance, delivers high scalability and balances load evenly across the nodes. The analysis ensures that the performance of our system scales beyond and is not an artifact of our measurements.

We evaluate Meridian through simulations as well as a deployment on PlanetLab [4]. Our simulations are parameterized by an extensive measurement study, in which we collected node-to-node round-trip latency measurements for 2500 Internet name servers (6.25 million node pairs). We use 500 of these nodes as targets, and the remaining 2000 as overlay nodes in our experiments.

Overall, this paper makes three contributions. First, it outlines a lightweight, scalable, and accurate system for keeping track of location-information for participating nodes. The system is simple, loosely-structured, and entails modest resources for maintenance. The paper shows how Meridian can efficiently find the closest node to a target, the latency minimizing node to a given set of nodes, and the set of nodes that lie in a region defined by latency constraints, which are frequently encountered building block operations in many location-sensitive distributed systems. Although less general than virtual coordinates, we show that Meridian incurs significantly less error. Second, the paper provides a theoretical analysis of our system that shows that Meridian provides robust performance, high scalability and good load balance. This analysis is general and applies to Internet latencies that cannot be accurately modeled with a Euclidean metric. Following a line of previous work on object location (see [24] for a recent summary), we give guarantees for the family of growth-constrained metrics. Moreover, we support a much wider family of metrics of low *doubling dimension*. Finally, the paper shows empirical results from both simulations parameterized with measurements from a large-scale network study and a PlanetLab deployment. The results confirm our theoretical analysis that Meridian is accurate, scalable, and load-balanced.

2. 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.

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. We later show in the analysis (Section 4) that a choice of $k = O(\log N)$ can resolve queries in $O(\log N)$ lookups; in simulations (Section 6), we verify that a small k suffices. We assume that every participating node has a rough estimate of $\log N$.

The ring structure with its exponentially increasing ring radii favors nearby neighbors, enabling 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.

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 provides much greater utility than a set of ring members that are clustered together, as shown in Figure 1. Intuitively, nodes that are geographically diverse instead of clustered together enable a node to forward a query to a greater region. 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 hy-

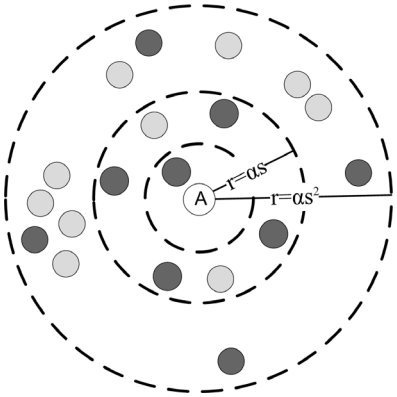

Figure 1: Each Meridian node keeps track of a fixed number of other nodes and organizes these nodes into concentric, non-overlapping rings of exponentially increasing radii.

pervolume, each node defines a local, non-exported coordinate space. A node i will periodically measure its distance d_j^i 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 $\langle d_1^i, d_2^i, ..., d_{k+l}^i \rangle$, 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 discovered 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 a routing query can reduce query performance; k can be increased to compensate for this expected rate of failure.

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. Our gossip protocol is based on an anti-entropy push protocol [17] 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 re-pinging 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.

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, and is 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.

3. APPLICATIONS

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

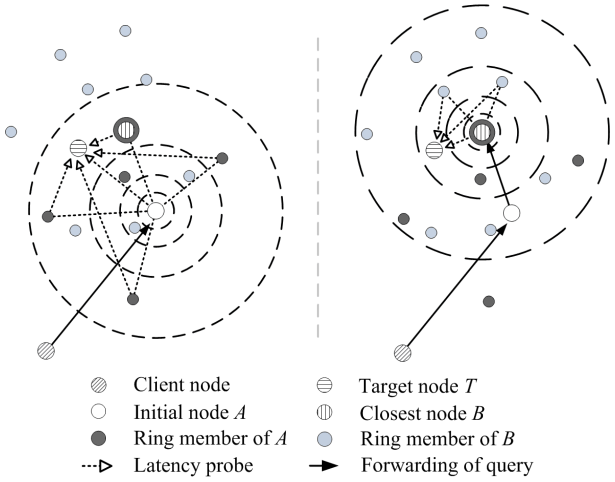

Figure 2: A client sends a "closest node discovery to target T" request to a Meridian node A, which determines its latency d to T and probes its ring members between $(1-\beta)\cdot d$ and $(1+\beta)\cdot d$ to determine their distances to the target. The request is forwarded to the closest node thus discovered, and the process continues until no closer node is detected.

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 in structured peer-to-peer networks such as Chord [55], Pastry [48] and Tapestry [61], 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 cannot be closer to the target than the Meridian node currently processing the query.

Meridian uses an acceptance threshold β, which determines 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 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 reduces error at the expense of increased hop count.

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. Typically, such a node plays a specialized role in the network that 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.

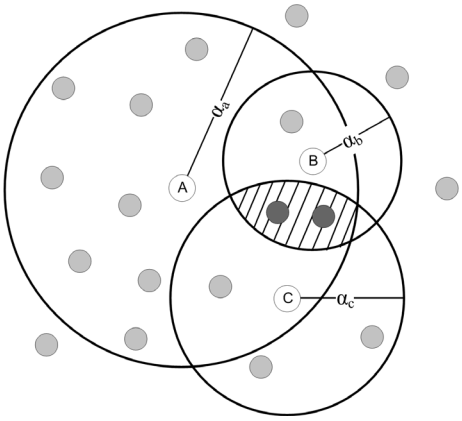

Figure 3: A multi-constraint query consisting of targets A, B, C with respective latency constraints of $\alpha_a, \alpha_b, \alpha_C$. The shaded area represents the solution space.

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 $d_{avg} = (\sum_{i=1}^{|T|} d_i)/|T|$. Similarly, when a ring member is requested to determine its latency to the targets, it computes the average latency and returns that to the requesting node. The remaining part of the central leader election application follows exactly from the closest node discovery protocol.

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 3. 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 $\langle \textit{target}_i, \textit{range}_i \rangle$, for all $0 < i \leq u$, it measures its latency d_i to the target nodes and calculates its distance to the solution space as

$$s = \sum_{i=1}^{u} \max(0, d_i - \textit{range}_i)^2$$

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 $\max(0, (1-\beta)\cdot(d_i - \textit{range}_i))$ to $(1+\beta)\cdot(d_i + \textit{range}_i)$ 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 $\max_{0<i\leq u}((2\beta+1)\cdot(d_i + \textit{range}_i))$ 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.

4. ANALYSIS

In this section we argue analytically that Meridian scales well with the size of the system. Our contributions are three-fold. First, we put forward a rigorous definition that captures the quality of the ring sets, and prove that under certain reasonable assumptions, small ring cardinalities suffice to ensure good quality. Second, we show that with these good-quality rings, the nearest-neighbor queries return exact or near-exact neighbors in logarithmic number of steps. Finally, we argue that if the ring sets of different nodes are stochastically independent then the system is load-balanced; that is, if many random queries are inserted into the system, then the load is spread approximately evenly among Meridian nodes.

We model the matrix of Internet latencies as a metric, that is, a symmetric function obeying the triangle inequality. Since one cannot hope to provide theoretical guarantees for arbitrary metrics, we will need to make some reasonable assumptions to capture the properties of real-life latencies. However, we avoid assumptions on the *geometry* of the metric. Most notably, we do not assume that the metric is Euclidean, since recent experimental results suggest that approximating Internet latencies by Euclidean metrics, although a useful heuristic in some cases, incurs significant relative errors [42, 15, 40, 57, 52, 44, 12, 43, 39].

We will consider two families of metrics that have been popular in the recent theoretical literature as non-geometric notions of low-dimensionality: *growth-constrained* metrics and *doubling* metrics. We focus on the case when the rate of churn and fluctuations in Internet latencies are sufficiently low to provide Meridian with time to adjust. So, for the purposes of this analysis, we assume that the node set and the latency matrix do not change with time.

Proofs of the forth-coming theorems are deferred to a complementary technical report [60]. We provide proof sketches below.

Preliminaries. Nodes running Meridian are called *Meridian nodes*. When such a node receives a query to find the nearest neighbor of some node q, this q is called the *target*. Let V be the set of all possible targets. Let $S_M \subset V$ be the set of Meridian nodes, of size N. Let d be the distance function on V induced by the node-to-node latencies: d_{uv} is the uv-distance, that is, the latency between nodes u and v. Let $B_u(r)$ denote the closed ball in S_M of radius r around node u, that is, the set of all Meridian nodes within distance r from u; let $B_{ui} = B_u(2^i)$. For simplicity let the smallest distance be 1 and denote the maximal distance by Δ.

For some fixed k, every node u maintains $\log(\Delta)$ *rings* $S_{ui} \subset B_{ui} \setminus B_{(u,\,i-1)}$ of exactly k nodes each; the elements of these rings are called *Meridian neighbors* of u. We treat each ring at a given time as a random variable whose values are k-node subsets of S_M. In particular, we can talk about the distribution of a given ring, and about several rings being probabilistically independent from each other.

Ring quality. Intuitively, we want each ring S_{ui} to cover the corresponding annulus $B_{ui} \setminus B_{(u,\,i-1)}$ reasonably well; that is, we want each node in this annulus to be within a relatively small distance from some Meridian neighbor of u. We say that Meridian rings are *ϵ-nice*, if for any two nodes $u, v \in S_M$, node u has a Meridian neighbor $w \in S_{ui}$ such that $d_{wv} \leq \epsilon\, d_{uv}$ and $d_{uv}(1+\epsilon) \leq 2^i$. If the Meridian rings are ϵ-nice then our algorithm finds good approximate nearest neighbors; the precision improves as ϵ gets smaller. Under some additional assumptions, even $\epsilon = \frac{1}{2}$ suffices to guarantee finding *exact* nearest neighbors.

We will show that even with small ring cardinalities it is possible to make the rings ϵ-nice, and confirm this with empirical evidence in Section 5. We give constructive arguments where we show that the rings with small cardinalities are ϵ-nice provided that the ring sets, seen as stochastic distributions over subsets of nodes, have certain reasonable properties.

4.1 Growth-constrained metrics

For n-dimensional grid and $\alpha = O(n)$, the cardinality of any ball is at most 2^α times smaller than the cardinality of a ball with the same center and twice the radius. This motivates the following definition: the *grid dimension* of a metric is the smallest α such that the above property holds. If the grid dimension is constant (and, intuitively, small), we say that the metric is *growth-constrained*. Growth-constrained metrics can be seen as generalized grids; they have been considered in the context of compact data structures [30], and have been used as a reasonable abstraction of Internet latencies (see [25] for a short survey).

We start with a model where the metric on the Meridian nodes is growth-constrained, but we make no such assumption about the non-Meridian nodes. This is important because even in an unfriendly metric we might be able to choose a relatively well-behaved subset of Meridian nodes.

Our first result is that even with small ring cardinalities it is possible to make the rings ϵ-nice. We say at some point of time the ring S_{ui} is *well-formed* if it is distributed as a random k-node subset of $B_{ui} \setminus B_{(u,\,i-1)}$. Intuitively, this is desirable since, in a growth-constrained metric, the density is approximately uniform.

Theorem 4.1 *Let the metric on S_M have grid dimension α. Fix $\delta \in (0,1)$ and $\epsilon < 1$; let the cardinality of a Meridian ring be $k = O(\frac{1}{\epsilon})^\alpha \log(N/\delta)$. If the Meridian rings are well-formed then with probability at least $1-\delta$ they are ϵ-nice.*

Proof Sketch: Fix two Meridian nodes u, v and let $r = \epsilon d_{uv}$. Pick the smallest i such that $d_{uv} + r \leq 2^i$. Then

$$B_{ui} \subset B_v(2^i + d_{uv}) \subset B_v(2^{i+1} - r) = B_v(\gamma r),$$

where $\gamma = 4 + 3/\epsilon$. By definition of the grid dimension $|B_{ui}| \leq \gamma^\alpha |B_v(r)|$. By Chernoff Bounds[2], some node from S_{ul}, $l \leq i$ lands in $B_v(r)$ with failure probability $< \delta/N^2$. □

[2]Chernoff Bounds establish that the sum of many bounded independent random variables is close to its expectation with very high probability.

Recall that our nearest-neighbor search algorithm forwards a query to the node $w \in S$ that is closest to the target t, subject to the constraint that $d_{ut}/d_{wt} \leq \beta_0$; if such w does not exist, the algorithm stops. Here $\beta_0 > 1$ is a parameter; we denote this algorithm by $\mathcal{A}(\beta_0)$.

Consider a node q and let u be its nearest neighbor. Say node v is a γ-*approximate* nearest neighbor of q if $d_{vq}/d_{uq} \leq \gamma$. An algorithm $\mathcal{A}(\beta_0)$ is γ-*approximate* if, for any query, it finds a γ-approximate nearest neighbor and does so in at most $2\log(\Delta)$ steps.

The following theorem describes the quality of algorithm $\mathcal{A}(\beta_0)$, for different values of the threshold β_0, under the assumption that the rings are ϵ-nice. In part (a) the algorithm looks at only three rings at every intermediate node; in parts (bc) it might need to look at $O(\log \frac{1}{\epsilon})$ rings. The tradeoff between β_0 and the approximation ratio matches our simulation in Section 6 (Figure 7).

Theorem 4.2 *Suppose the Meridian rings are ϵ-nice.*

(a) algorithm $\mathcal{A}(2)$ is 3-approximate if $\epsilon \leq \frac{1}{8}$;

(b) $\mathcal{A}(1+\epsilon^2)$ is $(1+3\epsilon)$-approximate if $\epsilon \leq \frac{1}{4}$;

(c) $\mathcal{A}(1+\gamma)$ is $(1+3\epsilon+\gamma)$-approximate if $\epsilon \leq \frac{1}{4}$, $\gamma \leq \frac{2}{5}$.

Proof Sketch: Let q be the target node and w be the nearest neighbor of q. For a node u, let $r(u) = d_{uq}/d_{wq}$ be the approximation ratio. If the query is forwarded from node u to node v, we say that the *progress* at u is d_{uq}/d_{vq}.

For part (a) we show that the progress is at least 2 at every node u such that $r(u) \geq 3$, so in at most $\log \Delta$ steps we reach some node v such that $r(v) < 3$.

For parts (bc) we define a function $f(x)$ which is continuously increasing from $f(1) < 1+3\epsilon$ to infinity, and show that algorithm $\mathcal{A}(\beta_0)$ achieves progress $x \geq \beta_0$ at any node u such that $r(u) = f(x)$. The query is thus forwarded from u to some node v within distance d_{uq}/x from q; it follows that $r(v) \leq f(x)/x$.

The query proceeds in two stages. In the first stage the progress at each node is $x \geq 2$; in at most $\log \Delta$ steps we reach some node u such that $r(u) < f(2)$. For the second stage, the progress can be less than 2. The crucial observation is that $f(1+y)/(1+y) \leq f(1+y/2)$ for any $y \leq 1$. Therefore if for the current node $r(\cdot)$ is $f(1+y)$, then for the next node it is at most $f(1+y/2)$.

If $\beta_0 = 1+\gamma$ then we can iterate this $\log \frac{1}{\gamma}$ times and reach a node such that $r(\cdot) \leq f(1+\gamma/2)$. For part (c) we just note that $f(1+\gamma/2) < 1+3\epsilon+\gamma$. For part (b) we take $\gamma = \epsilon^2$ and note that $f(1+\epsilon^2/2) \leq 1+3\epsilon$. □

In Thm. 4.1 the value of k depends on ϵ. We can avoid this and find exact nearest-neighbors by restricting the model. Specifically, we assume that the metric on $S_M \cup \{q\}$ is growth-constrained for any target q in some set $Q \subset V$. However, we do not need to assume that the metric on *all* of Q is growth-constrained; in particular, very dense clusters of targets are allowed.

We modify $\mathcal{A}(\beta_0)$ slightly: if w is the Meridian neighbor of the current node u that is closest to the target t, and $d_{ut}/d_{wt} < \beta_0$, then instead of stopping at u the algorithm stops at w. Denote this modified algorithm by $\mathcal{A}'(\beta_0)$; say it is Q-*exact* if it finds an exact nearest neighbor for all queries to targets in the set Q, and does so in at most $\log(\Delta)$ steps.

Theorem 4.3 *Fix some set $Q \subset V$ such that for any $q \in Q$ the metric on $S_M \cup \{q\}$ has grid dimension α. Fix $\delta \in (0,1)$ and let $k = 2^{O(\alpha)} \log(N|Q|/\delta)$ be the cardinality of a Meridian ring. If the rings are well-formed then with probability at least $1-\delta$ algorithm $\mathcal{A}'(2)$ is Q-exact.*

Proof Sketch: Using the technique from Thm. 4.2a, we prove that the distance to target decreases by a factor of at least 2 on each step except maybe the last one. We have to be careful about this last step, since in general the target is not a Meridian node and therefore not a member of any ring. In particular, this is why we need bounded grid dimension on $S_M \cup \{q\}$, not only on S_M. □

Ideally, the algorithm for nearest neighbor selection would balance the load among participating nodes. Intuitively, if $N_{\text{qy}}(\mathcal{A})$ is the maximal number of packets exchanged by a given algorithm $\mathcal{A}$ on a single query, then for m random queries we do not want any node to send or receive much more than $\frac{m}{n} N_{\text{qy}}(\mathcal{A})$ packets.

We make it precise as follows. Fix some set $Q \subset V$ and suppose each Meridian node u receives a query for a random target $t_u \in Q$. Say algorithm $\mathcal{A}$ is (γ, Q)-*balanced* if, in this scenario under this algorithm, any given node sends and receives at most $\gamma N_{\text{qy}}(\mathcal{A})$ packets. To reason about load balance, we need a slightly more restrictive model in which the metric on all of Q is growth-constrained and the rings are stochastically independent from each other. The latter property matches well with our simulation results (Figure 10).

Theorem 4.4 *Fix some set $Q \subset V$ such that the metric on Q has grid dimension α. Let S_M be a random N-node subset of Q. Fix $\delta \in (0,1)$ and let the cardinality of a Meridian ring be $k = 2^{O(\alpha)} \log(|Q|/\delta) \log(N) \log(\Delta)$.*

If the rings are well-formed and stochastically independent then with probability at least $1-\delta$ algorithm $\mathcal{A}'(2)$ is Q-exact and (γ, Q)-balanced, for $\gamma = 2^{O(\alpha)} \log(N\Delta/\delta)$.

Proof Sketch: We prove that $\mathcal{A}'(2)$ is Q-exact using the technique from Thm. 4.3. Some extra computation is needed due to the fact that we do not have a good bound on the grid dimension of S_M; instead, we are given that S_M is a random subset of Q.

The load-balancing property is much harder to prove, essentially because we need to bound, over all nodes, not only the expected load (which is relatively easy), but also the actual load. We consider the probability space where the randomness comes from choosing Meridian nodes, Meridian neighbors, and the query targets t_u, $u \in S_M$. In this space, we consider the N nearest-neighbor queries propagating through the Meridian network. Ideally, we would like to express the contribution of a given query i to the load on a given node u as a random variable L_{ui}, and use Chernoff Bounds to show that with high probability the sum $\sum_i L_{ui}$ does not deviate too much from its expectation. However, Chernoff Bounds only apply to *independent* random variables, which the L_{ui}'s are not. To remedy this, we need to be a lot more careful in splitting the load on u into a sum of random variables; see the full version [60] for details. □

Extensions. We can further show that if a metric is comparatively 'well-behaved' in the vicinity of a given node, then some of its rings can be made smaller. Second, our guarantees are worst-case; *on average* it suffices to query only a fraction of Meridian neighbors of a given ring, e.g. $(\frac{1}{\epsilon})^{O(\alpha)}$ neighbors in Thm. 4.1. Third, our results for growth-constrained metrics hold under a less restrictive definition of grid dimension that only applies to balls of cardinality at least $\log N$. Finally, Thm. 4.4 holds under a more demanding definition of (γ, Q)-balanced algorithm. We discuss these extensions in the full version [60].

4.2 Doubling metrics

Any point set in an n-dimensional Euclidean metric has the following property: for $\alpha = O(n)$, every ball of radius r can be

covered by 2^α balls of radius $r/2$. For an arbitrary metric, we define the *doubling dimension* [21] as the smallest α such that the above property holds. If the doubling dimension is constant (and, intuitively, small), we say that the metric is *doubling*. Such metrics have recently become an active research topic [21, 56, 37, 32].

By definition, doubling metrics generalize low-dimensional Euclidean metrics. This generalization is non-trivial: by [21] there exist doubling metrics on N nodes that need distortion $\Omega(\sqrt{\log N})$ to embed into any Euclidean space. It is known [21] that the doubling dimension is at most four times the grid dimension, so doubling metrics also generalize growth-constrained metrics. Doubling metrics are more powerful because they can combine very sparse and very dense regions, e.g. for the *exponential line*, the subset $\{2^i : 0 \leq i \leq N\}$, the doubling dimension is 1, but the grid dimension is $\log N$.

For doubling metrics, the notion of well-formed rings is no longer adequate, since we might need to boost the probability of selecting a node from a sparser region. In fact, this is precisely the goal of our ring-membership management in Section 2. Fortunately, mathematical literature provides a natural way to formalize this intuition.

Say a measure is *s-doubling* [23] if for any ball B, the measure of B is at most s times larger than that of a ball with the same center and half the radius. Intuitively, a doubling measure is an assignment of weights to nodes that makes a metric look growth-constrained. It is known [23] that for any metric of doubling dimension α there exists a $2^{O(\alpha)}$-doubling measure μ.

Say that at some point of time the ring S_{ui} is *μ-well-formed* if it is distributed as a random k-node subset of $S = B_{ui} \setminus B_{(u,\,i-1)}$, where nodes are drawn with probability $\mu(\cdot)/\mu(S)$. Using these notions, one can obtain the guarantee in Thm. 4.1, where, instead of the grid dimension, we plug in a potentially much smaller doubling dimension of S_M.

Theorem 4.5 *Suppose the metric on S_M has doubling dimension α, and let μ be a 2^α-doubling measure on S_M. Fix $\delta \in (0,1)$ and $\epsilon \leq 1$; let $k = O(\frac{1}{\epsilon})^\alpha \log(N/\delta)$ be the cardinality of a Meridian ring. If the Meridian rings are μ-well-formed, then with probability at least $1-\delta$ they are ϵ-nice. In particular, Thm. 4.2 applies.*

5. EVALUATION

We evaluated Meridian through both a large scale simulation parameterized with real Internet latencies and a physical deployment on PlanetLab.

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 PlanetLab 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 [20].

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.

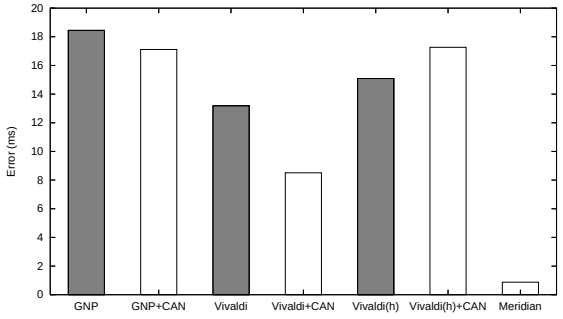

Figure 4: Light bars show the median error for discovering the closest node. Darker bars show the inherent embedding error with coordinate systems. Meridian's median closest node discovery error is an order of magnitude lower than schemes based on embeddings.

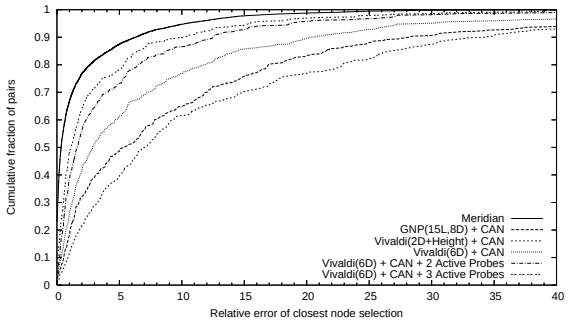

Figure 5: Meridian's relative error for closest node discovery is significantly better than virtual coordinates.

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 [15]. 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 bars in Figure 4 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 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 4, are an order of magnitude higher than Meridian. Figure 5 compares the relative error CDFs of different closest node discovery schemes. Meridian

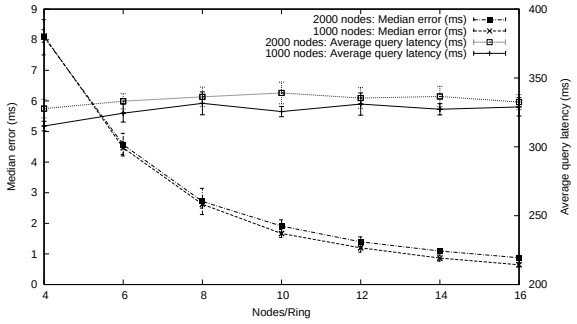

Figure 6: A modest number of nodes per ring achieves low error. Average latency is determined by the slowest node in each ring and the hop count, and remains constant within measurement error bounds.

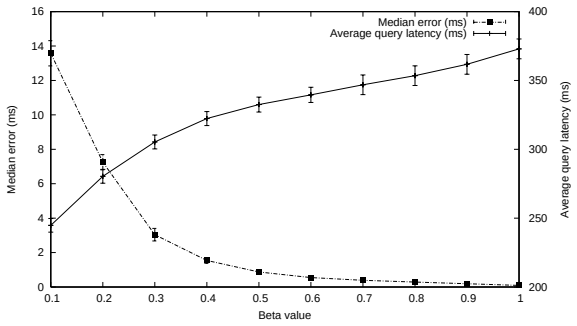

Figure 7: An increase in β significantly improves accuracy for $\beta \leq 0.5$. The average query latency increases with increasing β, as a bigger β increases the average number of hops taken in a query.

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. Figure 5 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. 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 6 shows that median error drops sharply as k increases. Hence, a node 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 initial 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

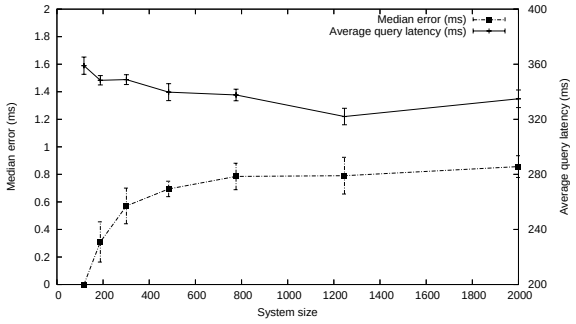

Figure 8: Median error and average query latency as a function of system size, for $k = \lfloor \log_{1.6} N \rfloor$; both remain constant as the network grows, as predicted by the analytical results.

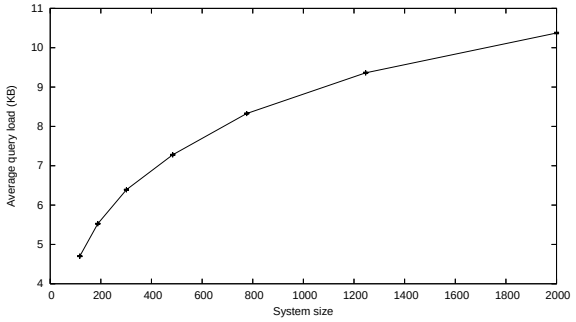

Figure 9: The average load of a closest node discovery query increases sub-linearly with system size ($k = \lfloor \log_{1.6} N \rfloor$).

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 more time cannot be a closer node and its result is discarded. The average query latency curve in Figure 6 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.

The β parameter captures the tradeoff between query latency and accuracy as shown in Figure 7. 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 $\beta \leq 0.5$; this matches our analysis (see Thm. 4.2). 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 8 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. As predicted by the theoretical analysis in Section 4, 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 network, as this can limit the number of concurrent closest node discoveries that can be performed at a particular system size.

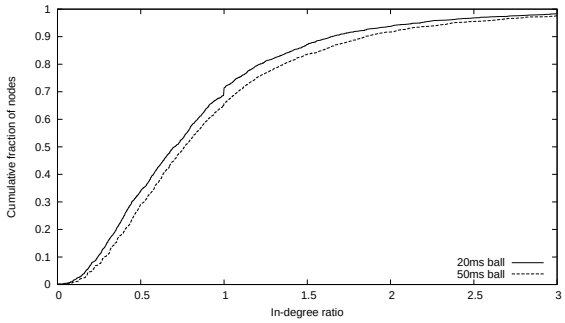

Figure 10: The in-degree ratio shows the average imbalance in incoming links within spherical regions. More than 90% of regions have a ratio less than 2.

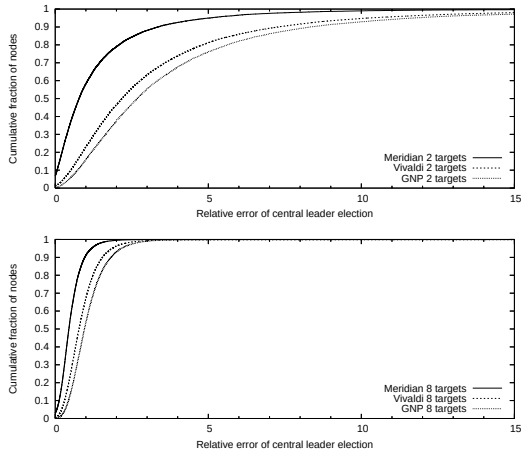

Figure 11: Central leader election accuracy.

Figure 9 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, and shows that it grows sublinearly with system size, with 2000 nodes requiring a total of 10.4 KB per query.

A desirable property for load-balancing, and one of the assumptions in our theoretical analysis (see Thm. 4.4) 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 10 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, as well as an assumption in our theoretical analysis (see Thm. 4.2), 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

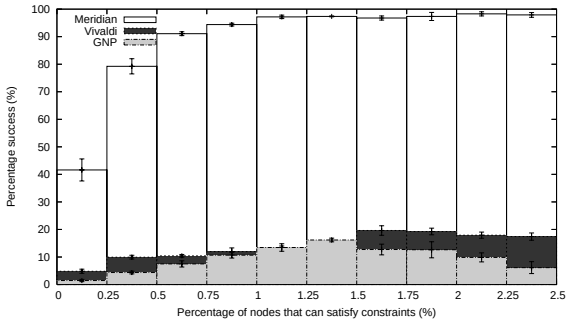

Figure 12: The percentage of successful multi-constraint queries is above 90% when the number of nodes that can satisfy the constraints is 0.5% or more.

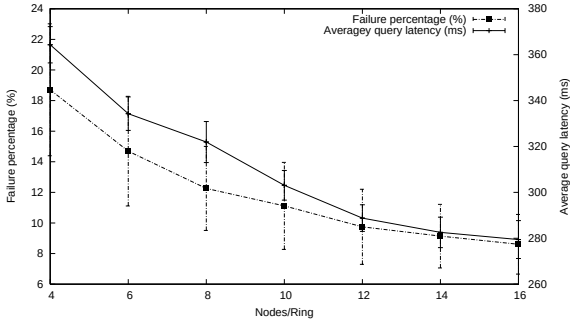

Figure 13: An increase in the number of nodes per ring k significantly reduces the failure percentage of multi-constraint queries for $k \leq 8$.

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 11 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 constraint 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 12 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.

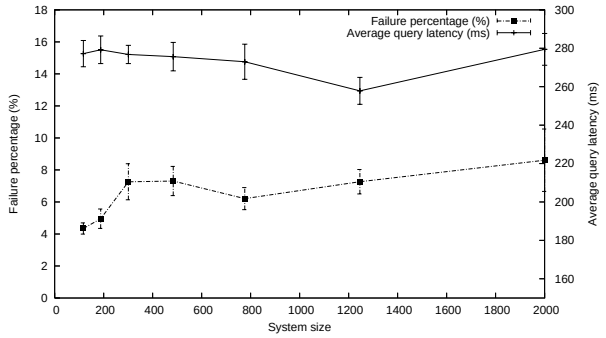

Figure 14: The percentage of multi-constraint queries that cannot be resolved with Meridian and average query latency. Both are independent of system size.

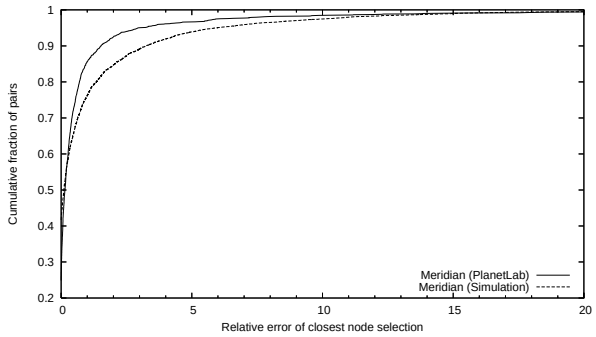

Figure 15: The relative error of closest node discovery for a Meridian deployment on PlanetLab versus simulation. Meridian achieves results comparable to or better than our simulations in a real-world deployment.

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 13 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 14 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.

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 1.8ms, and the relative error CDF in Figure 15 shows that it performs better than simulation results from a similarly configured system.

6. RELATED WORK

Meridian is a general network location service that we have applied to three separate location-related problems. We separate past work on these problems into approaches that rely on network embeddings and those that do not, and survey both in turn.

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 [42] determines the coordinates of a node by measuring its latency to a fixed set of landmarks and then solving a multidimensional, iterative, nonlinear minimization problem. ICS [40] and Virtual Landmarks [57] 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. To avoid the load imbalance and lack of failure resilience created by a set of fixed landmarks, PIC [12] and PCoord [39] 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 [43] 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 [44] 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 [15] is based on a simulation of springs, and can be augmented with an additional height vector to increase accuracy. Big-Bang Simulation [52] performs a simulation of a particle explosion under a force field to determine node positions.

IDMaps [19] 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 [18] has also examined how to delegate probing to specialized nodes in the network.

Recent theoretical work [32, 54] has sought to explain the empirical success of network embeddings and IDMaps-style approaches.

Server Selection: Our closest node discovery protocol draws its inspiration from the Chord DHT [55], which performs routing in a virtual identifier space by halving the virtual distance to the target at each step. Proximity based neighbor selection [9, 8] 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 [27] and [30], 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 [33], 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 [47] operates similarly, using approximate bin numbers instead of direct latency measurements. Mithos [58] 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 [51]. These schemes require $O(N)$ state per node, which limits their scalability, and are non-trivial to adapt to other positioning problems. Tiers [3] reduces the state requirement by forming a proximity-aware tree and performing a top-down search to discover the closest node. Hierarchical systems 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 [22] 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 [28]. Dynamic server selection was found in [6] 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 [7] 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.

7. CONCLUSIONS

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 paper, 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 argued analytically that Meridian provides robust performance, delivers high scalability, and balances load evenly across nodes. 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 shown how the framework can be used to solve three network positioning problems frequently-encountered in distributed systems; it remains to be seen whether the lightweight approach advocated in this paper can be applied to other significant problems.

Acknowledgments

We would like to thank our shepherd, Jon Crowcroft, Jon Kleinberg and the anonymous reviewers for many helpful comments and suggestions. We are grateful to Frank Dabek, Russ Cox, Frans Kaashoek, Robert Morris, Eugene Ng and Hui Zhang for sharing with us the Vivaldi and GNP software and data.

8. REFERENCES

[1] D. Andersen, H. Balakrishnan, M. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Symposium on Operating Systems Principles*, Banff, AB, Canada, October 2001.

[2] P. Assouad. Plongements Lipschitziens dans R^n. *Bull. Soc. Math. France*, 111(4), 1983.

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

[4] 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.

[5] A. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting Scalable Multi-Attribute Range Queries. In *SIGCOMM*, Portland, OR, August 2004.

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

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

[8] 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.

[9] 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.

[10] U. G. Center. QHull. UIUC Geometry Center, QHull Computational Geometry Package, http://www.qhull.org, 2004.

[11] Y. Chu, S. Rao, and H. Zhang. A Case for End System Multicast. In *SIGMETRICS*, Santa Clara, CA, June 2000.

[12] 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.

[13] A. Crainiceanu, P. Linga, J. Gehrke, and J. Shanmugasundaram. Querying Peer-to-Peer Networks Using P-Trees. In *Intl. Workshop on the Web and Databases*, Paris, France, June 2004.

[14] W. Cui, I. Stoica, and R. Katz. Backup Path Allocation based on a Correlated Link Failure Probability Model in Overlay Networks. In *Intl. Conference on Network Protocols*, Paris, France, November 2002.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] A. Gupta, R. Krauthgamer, and J. Lee. Bounded Geometries, Fractals, and Low-Distortion Embeddings. In *Symposium on Foundations of Computer Science*, Cambridge, MA, October 2003.

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

[23] J. Heinonen. Lectures on Analysis on Metric Spaces. Springer Verlag, Universitext 2001.

[24] K. Hildrum, R. Krauthgamer, and J. Kubiatowicz. Object Location in Realistic Networks. In *Symposium on Parallelism in Algorithms and Architectures*, Barcelona, Spain, June 2004.

[25] K. Hildrum, J. Kubiatowicz, S. Ma, and S. Rao. A Note on Finding the Nearest Neighbor in Growth-Restricted Metrics. In *Symposium on Discrete Algorithms*, New Orleans, LA, January 2004.

[26] K. Hildrum, J. Kubiatowicz, and S. Rao. Another Way to Find the Nearest Neighbor in Growth-Restricted Metrics. In *UC Berkeley CSD ETR, UCB/CSD-03-1267*, UC Berkeley, August 2003.

[27] 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.

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

[29] K. Johnson, J. Carr, M. Day, and M. Kaashoek. The Measured Performance of Content Distribution Networks. In *Web Caching and Content Delivery Workshop*, Lisbon, Portugal, May 2000.

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

[31] D. Kempe, J. Kleinberg, and A. Demers. Spatial Gossip and Resource Location Protocols. In *Symposium on Theory of Computing*, Heraklion, Greece, July 2001.

[32] J. Kleinberg, A. Slivkins, and T. Wexler. Triangulation and Embedding using Small Sets of Beacons. In *Symposium on Foundations of Computer Science*, Rome, Italy, October 2004.

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

[34] R. Krauthgamer and J. Lee. The Intrinsic Dimensionality of Graphs. In *Symposium on Theory of Computing*, San Diego, CA, June 2003.

[35] R. Krauthgamer and J. Lee. Navigating Nets: Simple Algorithms for Proximity Search. In *Symposium on Discrete Algorithms*, New Orleans, LA, January 2004.

[36] R. Krauthgamer and J. Lee. The Black-Box Complexity of Nearest Neighbor Search. In *Intl. Colloquium on Automata, Languages and Programming*, Turku, Finland, July 2004.

[37] R. Krauthgamer, J. Lee, M. Mendel, and A. Naor. Measured Descent: A New Embedding Method for Finite Metrics. In *Symposium on Foundations of Computer Science*, Rome, Italy, October 2004.

[38] R. Lawrence. Running Massively Multiplayer Games as a Business, March 2004. Keynote speech from Networked Systems Design and Implementation.

[39] 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.

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

[41] P. Maniatis, M. Roussopoulos, T. Giuli, D. Rosenthal, M. Baker, and Y. Muliadi. Preserving Peer Replicas by Rate-Limited Sampled Voting. In *Symposium on Operating Systems Principles*, Bolton Landing, NY, October 2003.

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

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

[44] 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.

[45] C. Plaxton, R. Rajaraman, and A. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In *Symposium on Parallel Algorithms and Architectures*, Newport, RI, June 1997.

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

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

[48] 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.

[49] 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.

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

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

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

[53] A. Slivkins. Distance Estimation and Object Location via Rings of Neighbors. In *Symposium on Principles of Distributed Computing*, Las Vegas, NV, July 2005.

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

[55] 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.

[56] K. Talwar. Bypassing the Embedding: Approximation Schemes and Compact Representations for Growth Restricted Metrics. In *Symposium on Theory of Computing*, Chicago, IL, June 2004.

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

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

[59] H. Weatherspoon, T. Moscovitz, and J. Kubiatowicz. Introspective Failure Analysis: Avoiding Correlated Failures in Peer-to-Peer Systems. In *Intl. Workshop on Reliable Peer-to-Peer Distributed Systems*, Osaka, Japan, October 2002.

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

[61] 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.