Meridian: A Lightweight Framework for Network Positioning 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

February, 2005

Abstract

Selecting nodes based on their position in the network is a basic building block for many distributed systems. This paper describes a peer-to-peer overlay network for performing position-based node selection. Our system, Meridian, provides a lightweight, accurate and scalable framework for keeping track of location information for participating nodes. 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 in large-scale distributed systems without having to compute absolute coordinates; namely, closest node discovery, central leader election, and locating nodes that satisfy target latency constraints. We show analytically that the framework is scalable with logarithmic convergence when Internet latencies are modeled as a growth-constrained metric, a low-dimensional Euclidian metric, or a metric of low doubling dimension. Large scale simulations, based on latency measurements from 6.25 million node-pairs, and an implementation deployed on PlanetLab both show that the framework is accurate and effective.

1 Introduction

A central problem in distributed systems is to find an efficient mapping of system functionality onto nodes based on network characteristics. 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, service discovery, and multi-player online games, could substantially benefit 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 Euclidian space. For instance, recent work in network positioning [40, 12, 36, 52, 48, 41, 9, 39, 35] maps a large vector of node-to-node latency measurements on the Internet into a single point in a d-dimensional space. The resulting embedded address facilitates location-aware node selection.

While this approach is quite general, it is neither accurate nor complete. The embedding process typically introduces significant errors. The selection of parameters, such as the constant d and the set of measurements taken to perform the embedding, is nontrivial and has a significant impact on the accuracy of the approach. Coordinates change over time due to changes in network latencies on the Internet, and introduce additional errors when performing latency estimates from coordinates computed at different times. Finally, finding a set of nodes that match desired criteria without centralized servers that retain $O(N)$ state, an essential requirement in large-scale networks, requires additional mechanisms besides virtual coordinates. Peer-to-peer substrates that can naturally work with Euclidian coordinates, such as CAN [43] and P-Trees [10], can reduce the state requirements per node; however, both 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 a set of network positioning constraints. Meridian is based on a loosely-structured overlay network, uses direct measurements instead of a network embedding, and can solve spatial queries without an absolute coordinate space. It is similar in functionality to GNP combined with CAN in performing node selection based on network location[1].

Meridian is lightweight, scalable and accurate. Each Meridian node keeps track of a fixed number of peers and organizes them into concentric rings of exponentially increasing radii. A diverse node selection protocol is used in determining ring membership to maximize the marginal utility provided by each ring member. 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 the peer set. A scalable gossip protocol is used to notify other nodes of membership in the system. 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 further reduces errors stemming from out of date coordinates.

In this paper, we focus on three commonly-encountered network positioning problems in distributed systems, and describe how the lightweight Meridian framework can be used to resolve them without computing virtual coordinates. The first, and most significant, problem is that of discovering the closest node to a targeted reference point. This is a basic operation in content distribution networks (CDNs) [25], large-scale multiplayer games [34], and peer-to-peer overlays [24, 26, 6, 5]. Having the closest node serve the client or operate on the target can significantly reduce response time and aggregate network load. 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. Similarly, CDNs take network position into account when assigning clients to servers. And multiplayer games often perform a similar mapping from clients to nearby servers. In fact, the closest node discovery problem is so pervasive and so significant that we examine it in great detail. We also show that the Meridian framework can be used to find a node that offers minimal latencies to a given set of nodes (we make precise this notion of closeness in Section 3). Intuitively, we want to select a node that is at the centerpoint of the region defined by the set members. This basic operation can be used, for instance, for location-aware leader election, where it would enable the chosen leader to minimize the average communication latency from the leader to set members. Such an operation can be used in tree construction for an application-level multicast system, where it can reduce 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 that can sell node clusters with specific inter-node 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 is not an artifact of our measurements.

We evaluate Meridian through simulation parameterized by a large-scale network measurement study, and through a deployment on PlanetLab [2]. For our measurement study, we collected node-to-node round-trip latency measurements for 2500 nodes and 6.25 million node pairs on the Internet using the King [17] measurement technique. We use 500 of these nodes as targets, and the remaining 2000 as the 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, 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 [21] 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* which has recently become popular in the theoretical literature. Finally, the paper shows empirical results from both simulations using measurements from a large-scale network study and a PlanetLab deployment. The results confirm our theoretical analysis that Meridian is accurate, scalable,

[1] We use the term "location" to refer to a node's position in the Internet as defined by its roundtrip 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.

and load-balanced.

The rest of this paper is structured as follows. The next section describes the design and operation of the general-purpose Meridian framework. Section 3 illustrates how the target applications can be implemented on top of this framework. Section 4 contains a theoretical analysis of our closest node discovery protocol. Section 5 presents results from a large-scale Internet measurement study and evaluates the system based on real Internet data as well as a PlanetLab deployment. Section 6 discusses related work and Section 7 summarizes our findings.

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 $r_0 = 0$ 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. There is an upper limit of k on nodes kept in each ring, where peers are dropped 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 the maximum size of the system.

The rationale for exponentially increasing ring radii stems from the need for a node to have a representative set of pointers to the rest of the network, and the higher marginal utility nearby peers offer over faraway ones. The ring structure 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.

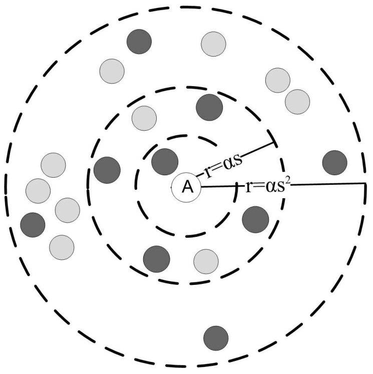

Figure 1: Each Meridian node keeps track of a fixed number of other nodes and organizes these nodes into concentric, non-overlapping rings of exponentially increasing radii. Within a given ring, a set of nodes that span a large amount of space (dark) are more desirable than a more limited subset (light).

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 has a significant effect on the performance of the system. For instance, if the nodes within a given ring are clustered together, their marginal utility is very small, despite their additional cost. A key principle then, as shown in Figure 1, is 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_j^i to another node j in its ring, for all $0 \leq i, j \leq k + l$. The coordinates of node i consist of the tuple $< d_1^i, d_2^i, ..., d_{k+l}^i >$, 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, a Meridian node can 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, we take a simple, greedy approach: 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 [7]. 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 [14] that implements a membership service. The central goal of our gossip protocol is not for each node to discover every node in the system, but simply for each node to discover a sufficiently diverse set of other nodes.

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 can be replaced as latency between nodes on the Internet changes dynamically. The newly discovered nodes are placed on B's rings as secondary members, subject to FIFO replacement.

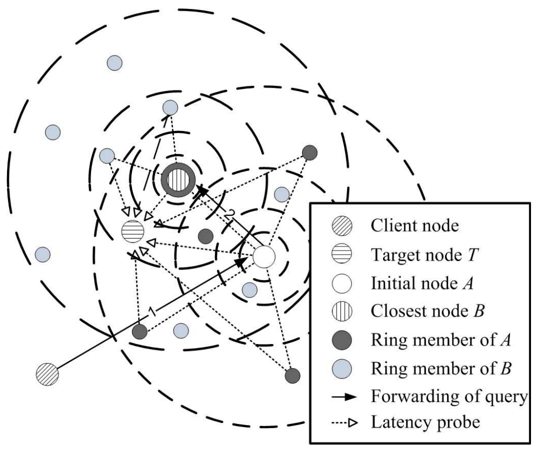

Figure 2: A client sends a "closest node discovery to target T" request to a Meridian node A, which determines its latency d to T and probes its ring members between $\frac{d}{2}$ and $\frac{3d}{2}$ to determine their distances to the target. The request is forwarded to the closest node thus discovered, and the process continues until no closer node is detected.

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.

3 APPLICATIONS

The following three sections describe how Meridian can be used to solve some common network positioning problems.

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 [50], Pastry [45] and Tapestry [54], 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

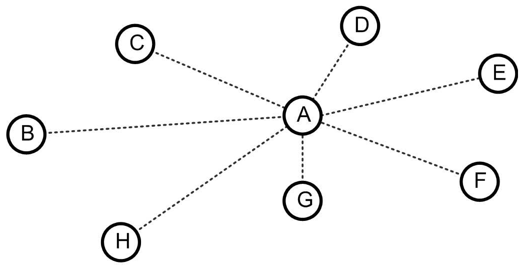

Figure 3: Central leader election selects the node with the minimum average distance to the nodes in the node group. For this example, node A is the best leader out of nodes A to H.

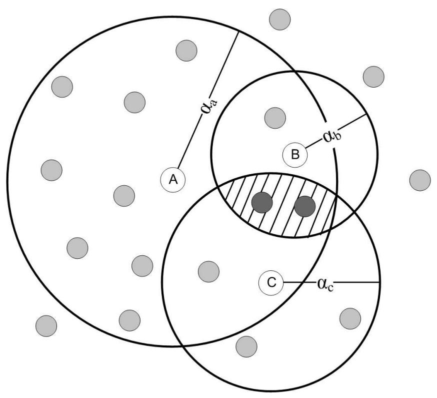

Figure 4: A multi-constraint query consisting of targets A, B, C with respective latency constraints of $\alpha_a, \alpha_b, \alpha_C$. The shaded area represents the solution space, and contains two nodes.

virtual identifier space. Another important distinction that Meridian holds over the structured peer-to-peer networks is the target nodes need not be part of the Meridian overlay. The only requirement is that the latency between a node on the overlay and a target node can be measured. 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 client request to find the closest node to a target, it determines the latency d between itself and the target. Once the latency is determined, it locates its corresponding ring j and simultaneously queries all nodes in that ring, as well as all nodes in the adjacent rings $j-1$ and $j+1$ whose distances to the origin are within $\frac{d}{2}$ to $\frac{3d}{2}$. These nodes measure their distance to the target and report the result back to the source. Nodes that take more than $2d$ to provide an answer are ignored, as they cannot be closer to the target than the source.

Meridian uses an acceptance threshold β, which serves a purpose similar to the routing base in structured peer-to-peer systems; namely, it 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 smaller 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 larger β may reduce 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 as illustrated in Figure 3. 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.

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} in the multi-target protocol. 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. An geographically distributed ISP may have thousands of nodes at its disposal, and finding the right set of nodes that satisfy the given constraints is necessary for satisfying the 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, $latency(q, p) < \alpha$ for some desired latency α.

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 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 $< target_i, range_i >$, 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

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

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 $max(0, \frac{d_i - \text{range}_i}{2})$ to $\frac{3 \cdot (d_i + \text{range}_i)}{2}$ 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 $2\max_i(d_i + \text{range}_i)$ for $0 < i \leq u$ to provide an answer are ignored.

The distance s_j of each node j to the solution space is calculated using (1). 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 hop count.

4 Analysis of scalability

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 work well, i.e. 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 the Meridian nodes.

We model the matrix of Internet latencies as a metric. We should not hope to achieve theoretical guarantees for *arbitrary* metrics; we need some reasonable assumptions to capture the properties of real-life latencies. We avoid assumptions on the *geometry* of the metric (e.g. we do not assume it is Euclidean) for two reasons. Firstly, recent experimental results suggest that approximating Internet latencies by Euclidean metrics, although a useful heuristic in some cases, incurs significant relative errors [40,12,36,52,48,41,9,39,35]. Secondly, and perhaps more importantly, even if we assume that the metric is Euclidean our algorithm is not allowed to use the coordinates – since one of the goals of this work is precisely to avoid heavy-weight embedding-based approaches.

We will consider two families of metrics that have been popular in the recent systems and theoretical literature as non-geometric notions of low-dimensionality: *growth-constrained* metrics and *doubling* metrics. In particular, growth-constrained metrics have been used as a reasonable abstraction of Internet latencies in the analysis of the object-location algorithm of Plaxton et al. [42]. Using a more general family of doubling metrics leads to good guarantees even for metrics that combine very dense and very sparse regions.

We focus on the case when the rate of churn and fluctuations in Internet latencies is sufficiently low so that Meridian has ample time to adjust. So for the purposes of this analysis we assume that the node set and the latency matrix are not changing with time.

Full proofs of the following theorems are quite detailed; they are deferred to Appendix A.

Preliminaries. Nodes running Meridian are called *Meridian nodes*. When such 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: denote the uv-distance by d_{uv}. Let $B_u(r)$ denote the closed ball in S_M of radius r around node u, i.e. 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; denote the maximal distance by Δ.

For some fixed k, every node u maintains $\log(\Delta)$ *rings* $S_{ui} \subset B_{ui}$ of exactly k nodes each; the elements of the rings are called *neighbors*. We treat each ring at a given time as a random variable; in particular, we can talk about a distribution of a given ring, and about rings being probabilistically independent.

Quality of the rings. Intuitively, we want each ring S_{ui} to cover the corresponding ball B_{ui} reasonably well, e.g. we might want each node in B_{ui} to be within a small distance from some node in S_{ui}. Moreover, for load-balancing it is bad if many different queries pass through the same node, so, intuitively, it is desirable that the rings of different nodes are probabilistically independent from each other.

Say a pair uv of Meridian nodes *ϵ-nice* if node u has a neighbor w within distance $\epsilon\, d_{uv}$ from v, and, moreover, $w \in S_{ui}$ where $2^{i-1} < d_{uv}(1+\epsilon) \leq 2^i$; say the rings are *$\epsilon$-nice* if all pairs of Meridian nodes are ϵ-nice.

In Thm. 4.3 and Thm. 4.4a it suffices for the rings to be $\frac{1}{2}$-nice; for better precision in a more relaxed model of Internet latencies (see Thm. 4.1) we might need smaller values of ϵ.

We will show that even with small ring cardinalities it is possible to make the rings ϵ-nice; this is later confirmed by the empirical evidence in Section 5 (see Fig. 12). We give a constructive argument where we show that the rings with small cardinalities are ϵ-nice provided that the ring sets (seen as stochastic distributions) have certain reasonable properties.

4.1 Growth-constrained metrics

Define the Karger-Ruhl dimension (*KR-dimension*) α as the log of the smallest s such that the cardinality of any ball $B_u(r)$ is at most s times smaller that of $B_u(2r)$. Say the metric is *growth-constrained* [26] if α is constant.

Since for a k-dimensional grid the KR-dimension is $O(k)$, growth-constrained metrics can be seen as generalized grids; they have been used as a reasonable abstraction of Internet latencies in past work (see the intro of [22] for a short survey). Growth-constrained metrics have also been considered in the context of dimensionality in graphs [32] and spatial gossip [27].

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}. Intuitively, this is desirable since in a growth-constrained metric the density is moreless uniform.

Theorem 4.1 *Assume the rings are well-formed; let the metric on Meridian nodes have KR-dimension α. Fix $\delta < 1$ and $\epsilon \leq 1$; set $k = O(\frac{1}{\epsilon})^{\alpha} \log(N/\delta)$. Then with probability at least $1-\delta$ the rings are ϵ-nice.*

Recall that our nearest-neighbor search algorithm forward the 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 $d_{vq}/d_{uq} \leq \gamma$. Say $\mathcal{A}$ is *γ-approximate* if for any query it finds a γ-approximate nearest neighbor, and does so in at most $2\log(\Delta)$ steps.

Theorem 4.2 *If the rings are ϵ-nice, $\epsilon \leq 1/8$ then*
(a) $\mathcal{A}(2)$ is 3-approximate,
(b) $\mathcal{A}(\beta_0)$ is $(1+\epsilon)$-approximate, $\beta_0 = 1 + O(\epsilon^2)$.[2]
(c) if we use a larger threshold $\beta_0 = 1+\gamma$, $\gamma \in (\epsilon, \frac{1}{2})$ then $\mathcal{A}(\beta_0)$ is $(1+\epsilon+2\gamma)$-approximate .

Note the tradeoff between the threshold β_0 and accuracy of the queries, which matches our simulation in Section 6 (see Figure 8).

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'll 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'll need to modify $\mathcal{A}(\beta_0)$ slightly: if w is the neighbor of the current node u that is closest to the target t, and $d_{ut}/d_{wt} \in (1, \beta_0)$ 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 KR-dimension α. Fix $\delta > 0$, let $k = 2^{O(\alpha)} \log(N|Q|/\delta)$, and assume the rings are well-formed. Then with probability at least $1-\delta$ algorithm $\mathcal{A}'(2)$ is Q-exact.*

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.

We'll need a somewhat more restrictive model. In particular, we'll assume that the metric on all of Q is growth-constrained, and that the rings are stochastically independent from each other. The latter property matches well with our simulation results (see Figure 11).

Theorem 4.4 *Fix some set $Q \subset V$ such that the metric on Q has KR-dimension α. Suppose S_M is a random N-node subset of Q. Let $k = 2^{O(\alpha)} \log(|Q|/\delta) \log(N) \log(\Delta)$.*

[2]Here $\mathcal{A}(\beta_0)$ might need to look at $O(\log \frac{1}{\epsilon})$ rings at every node.

(a) If the rings are well-formed then with probability at least $1-\delta$ algorithm $\mathcal{A}'(2)$ is Q-exact.

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

Note that in Thm. 4.4 it does not suffice to assume that S_M is an arbitrary subset of Q, since in general a subset of a growth-constrained metric can have a very high KR-dimension.

4.2 Extensions

Our results allow several extensions. The proofs are omitted from this version of the paper.

(1) Our results hold under a less restrictive definition of KR-dimension that only applies to balls of cardinality at least $x = \log n$; moreover, we can take $x = \log(n|Q|)$ in Thm. 4.3, and $x = \log|Q|$ in Thm. 4.4.

(2) We 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. We'd like the size of S_{ui} to depend only on what happens in the corresponding ball B_{ui}. Specifically, for $r = \epsilon\, 2^{i-3}$ we let σ_{ui} be the ratio of $|B_{ui}|$ to the smallest $|B_v(r)|$ such that $d_{uv} \leq 2^i - r$; note that $B_v(r) \subset B_{ui}$ for any such v. Then in Thm. 4.1 it suffices to assume that the cardinality of each ring S_{ui} is at least $2.2\, \sigma_{ui} \ln(n^2/\delta)$.

(3) Our guarantees are worst-case; *on average* it suffices to query only a fraction of neighbors of a given ring. Recall that on every step in algorithm $\mathcal{A}(\beta_0)$ we look at a subset S of neighbors and forward the query to the node $w \in S$ that is closest to the target t subject to the constraint that the *progress* of w, defined as the ratio d_{ut}/d_{wt}, is at least β_0. For $\beta_0 \leq 2$, suppose instead we forward the query to an arbitrary progress-2 node in S if such node exists. It is easy to check that all our results for $\mathcal{A}(\beta_0)$ carry over to this modified algorithm. Moreover, in Thm. 4.2a (used in conjunction with Thm. 4.1 for $\epsilon = 1/8$) instead of asking all neighbors of a given ring at once, we can ask them in random batches of size $k_0 = O(1)^\alpha$; then in expectation one such batch will suffice. Therefore on average on every step (except maybe the last one) we'll use only k_0 randomly selected neighbors from a given ring. Similarly, we can take $k_0 = O(\frac{1}{\epsilon})^\alpha$ for Thm. 4.2bc (used in conjunction with Thm. 4.1), $k_0 = O(\sigma_{ui})$ for Extension (2) above, and $k_0 = O(1)^\alpha$ for Thm. 4.3 and Thm. 4.4a. We obtain similar improvements for Thm. 4.2 used with Thm. 4.5 for doubling metrics.

(4) Thm. 4.4b holds under a stronger definition of a (β, Q)-balanced algorithm which allows more general initial conditions. Specifically, fix some $\gamma \geq 0$ and $m \geq n \cdot \max(\gamma, 1)$, and choose a random partition of m into n summands s_u, $u \in S_M$, such that $\gamma \leq s_u \leq \beta m/n$ for each u. Suppose each Meridian node u receives queries for s_u random targets in Q. Say algorithm $\mathcal{A}$ is (β, γ, m, Q)-*balanced* if under this algorithm in this scenario any given node sends and receives at most $\beta N_{\text{qy}}(\mathcal{A})$ packets. Note that an algorithm is (β, Q)-balanced if and only if it is $(\beta, 1, n, Q)$-balanced.

4.3 Doubling metrics

Define the *doubling dimension* DIM as the log of the smallest s such that every ball $B_u(r)$ can be covered by s balls of radius $r/2$. Metrics of low doubling dimension is a strictly more general family than growth-constrained metrics since it is easy to see [18] that DIM is at most four times the KR-dimension, but the converse is not true, e.g. for a subset $\{2^i :\ 0 \leq i \leq n\}$ of the real line DIM $= 1$, but KR-dimension is 2^n. Intuitively, doubling metrics are more powerful because they can combine very sparse and very dense regions. Moreover, doubling metrics can be seen as a generalization of low-dimensional Euclidean metrics; it is known [18] that for any finite point set in a k-dimensional Euclidean metric DIM $= O(k)$.

Doubling dimension has been introduced in the mathematical literature (see [20]) and has recently become a hot topic in the theoretical CS community [18, 30, 33, 51, 28, 49]; in particular it was used to model Internet latencies in the context of distributed algorithms for embedding and distance estimation [28, 49].

For metrics of low doubling dimension, well-formed rings are no longer adequate since we 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. Mathematical literature provides a natural way to make this intuition precise.

Say a measure is s-*doubling* [20] if for any ball $B_u(r)$ its measure is at most s times larger than that of $B_u(r/2)$. It is known [20] that for any metric there exists a 2^{DIM}-doubling measure μ. Intuitively, a doubling measure is an assignment of weights to nodes that makes a metric look growth-constrained; in particular, for exponential line $\mu(2^i) = 2^{i-n}$. 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 B_{ui}, where nodes are drawn with probability $\mu(\cdot)/\mu(B_{ui})$. Using these notions, one can obtain the guarantee in Thm. 4.1 where instead of the KR-dimension we plug in a potentially much smaller DIM of S_M.

Theorem 4.5 *Suppose the metric on S_M has doubling dimension* DIM*, and let μ be a 2^{DIM}-doubling measure on S_M. Fix $\delta < 1$ and $\epsilon \leq 1$; set $k = O(\frac{1}{\epsilon})^{\text{DIM}} \log(N/\delta)$. If the rings are μ-well-formed, then Meridian rings are ϵ-nice, so Thm. 4.2 applies.*

5 EVALUATION

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

Simulation. A large scale measurement study of 2500 DNS servers was used to parameterize simulations for evaluating Meridian. For the study, we collected pair-wise round trip time measurements between 2500 servers, spanning approximately 6.25 million node pairs. The study was replicated 9 times from 9 different PlanetLab nodes across North America, with the median value of the 10 runs taken for the round-trip time of each pair of nodes. We verified that each server has an unique IP address to reduce the likelihood that more than one of the chosen servers are hosted on the same machine. The experiment took approximately 8 days to complete, as the query interarrival times were dilated, and queries themselves randomized, to avoid queuing delays at the DNS servers. The experiments were performed from May 5 to May 13, 2004.

We obtained the latency measurement between DNS servers on the Internet via the King measurement technique [17]. King works as follows: assuming that a node S wants to determine the latency between DNS server A and B, it first sends a name lookup request to A and measure distance AS. Next, a recursive name request is sent to A for a domain where B is the authoritative name server, which will cause A to contact B on the measuring machine's behalf. This request will yield the roundtrip time from the measuring node to B via A, that is $AS + AB$. By taking the difference between the two measured times, S can determine an approximate roundtrip time between A and B.

In the following experiments, each of the tests consist of 4 runs with 2000 Meridian nodes, 500 target nodes, $k = 16$ nodes per ring, 9 rings per node, size of the innermost ring $s = 2$, probe packet size of 50 bytes, $\beta = 0.5$, and $\alpha = 1$ms, for 25000 queries in each run. The results are presented either as the mean result of the $100000 = 4 \times 25000$ 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 must discover other nodes in the system through the gossip protocol.

We first evaluate how accurate Meridian is in finding the closest node to a given target compared to the embedding based approaches. We computed the coordinates for our 2500 node data set using GNP, Vivaldi and Vivaldi with height. GNP represents an absolute coordinate scheme based on static landmarks. We configured it for 15 landmarks and 8 dimensions as suggested by the GNP authors, and used the N-clustered-medians protocol for landmark

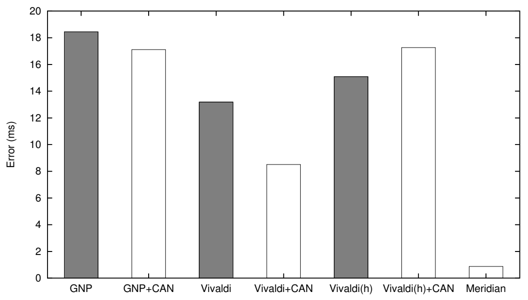

Figure 5: Light bars show the median error for discovering the closest node. Darker bars show the inherent embedding error with coordinate systems. Meridian's median closest node discovery error is an order of magnitude lower than schemes based on embeddings.

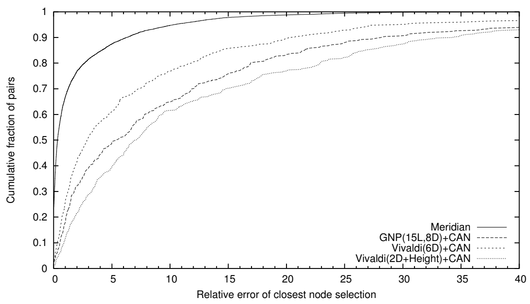

Figure 6: Meridian's relative error CDF for closest node discovery is significantly better than performing perfect query routing with an embedding scheme.

selection. Vivaldi is another absolute 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-Euclidian embedding which assigns a 2 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 selecting the closest nodes. The darker bars in Figure 5 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. However, even with a large embedding error, it is possible for the coordinate systems to pick the correct closest node. To evaluate this, we assumed the presence of a perfect geographic query routing layer, such as an actual 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 median closest node discovery error for all three embedding schemes, as shown by the lighter bars in Figure 5, are an order of magnitude higher than Meridian. Figure 6 compares the relative error CDFs of different closest node dis-

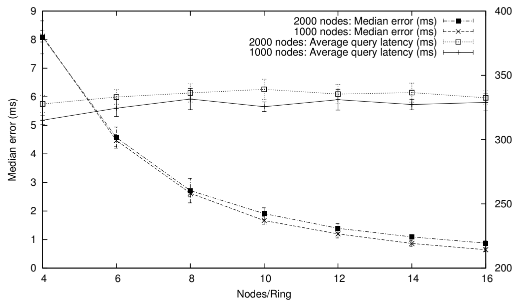

Figure 7: A modest number of nodes per ring achieves low error. Average latency is determined by the slowest node in each ring and the hop count, and remains constant within measurement error bounds.

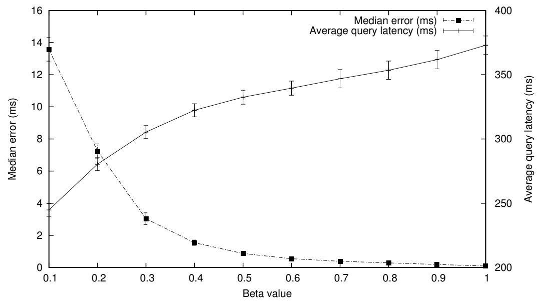

Figure 8: An increase in β significantly improves accuracy for $\beta \leq 0.5$. The average query latency increases with increasing β, as a bigger β increases the average number of hops taken in a query.

covery schemes. Meridian has a lower relative error than the embedding schemes by a large margin over the entire distribution.

The accuracy of our closest node discovery protocol depends on several parameters of our system, 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 this determines the granularity of the search where a higher number of nodes per ring will comb through the search space at a finer grain. Figure 7 shows the median error drops sharply as k increases. This is significant as 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. As each node only has nine total rings, a node must only be aware of at most seventy-two other nodes in the system.

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 processing overheads because they are negligible in comparison. Meridian bounds the maximum latency probe by two times the

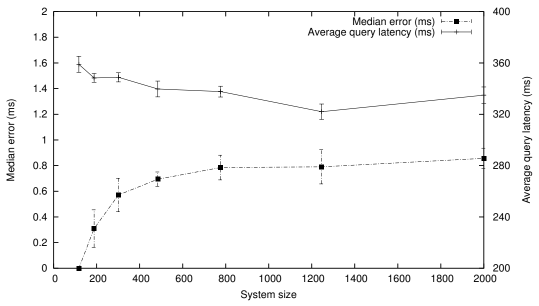

Figure 9: Error and query latency as a function of system size, for $k \sim \log N$. Both median error and average query latency remain constant as the network grows, as predicted by the analytical results.

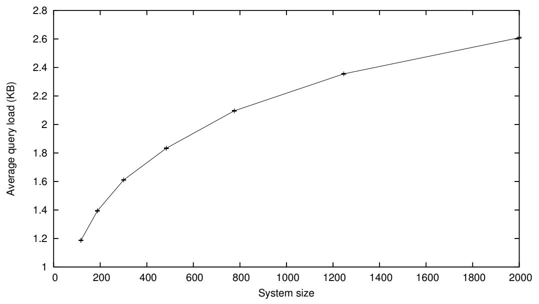

Figure 10: The average load of a closest node discovery query for $k \sim \log N$. The load increases sub-linearly with system size.

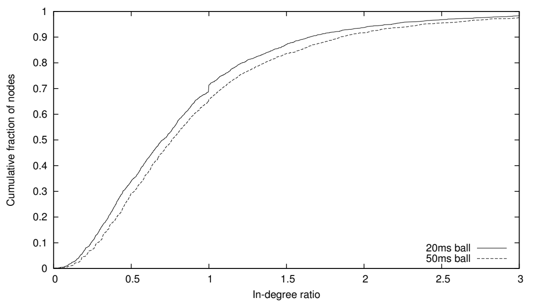

Figure 11: Load-balance in Meridian. The in-degree ratio shows the average imbalance in incoming links within spherical regions. More than 90% of regions have a ratio less than 2.

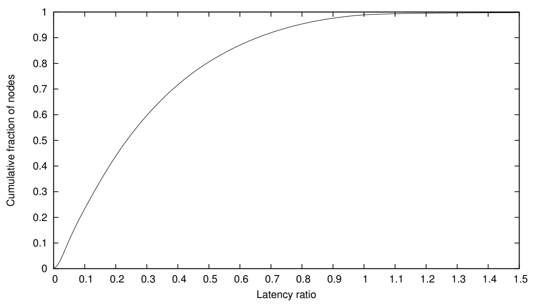

Figure 12: The efficacy of the ring member selection algorithm can be measured through the expected improvement ratio at each hop. For $\beta = 0.5$, Meridian's ring member selection algorithm can make progress via an extra hop to a closer node more than 80% of the time. For $\beta = 0.9$, an extra hop can be taken over 97% of the time.

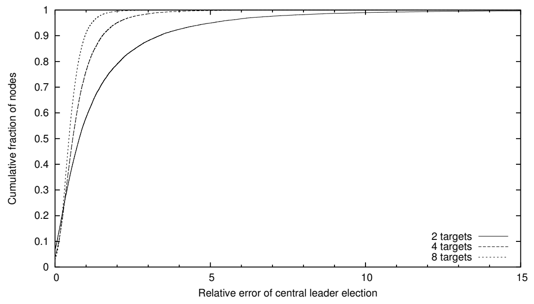

Figure 13: The accuracy of the central leader election application. With larger group sizes, the central leader election algorithm is able to find a centrally situated node more frequently.

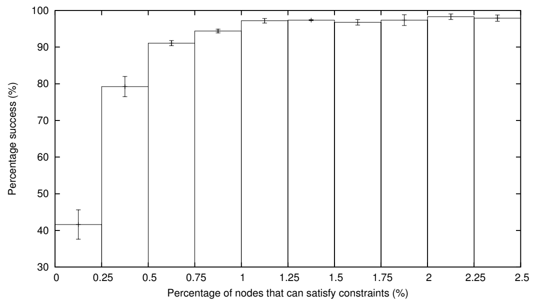

Figure 14: The percentage of successful multi-constraint queries is above 90% when the number of nodes that can satisfy the constraints is 0.5% or more.

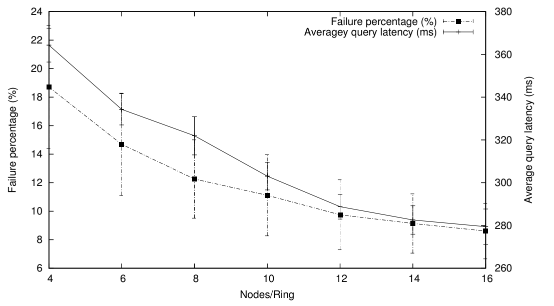

Figure 15: An increase in the number of nodes per ring k significantly reduces the failure percentage of multi-constraint queries for $k \leq 8$.

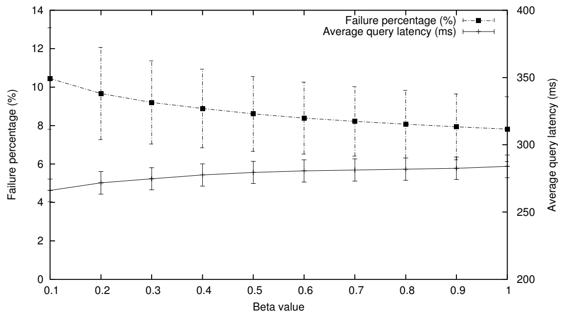

Figure 16: Increasing β decreases the failure percentage and increases the average latency of a multi-constraint query. Overall, the effect of β on application performance is small for multi-constraint node selection.

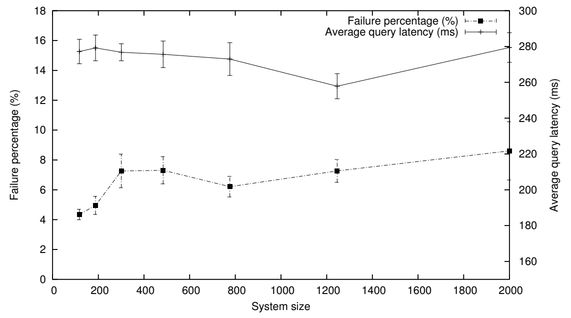

Figure 17: The percentage of multi-constraint queries that cannot be resolved with Meridian and average query latency. Both are independent of system size.

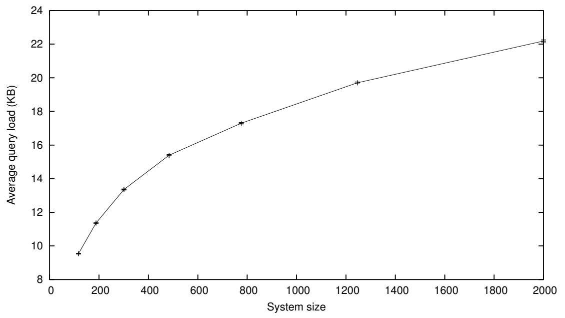

Figure 18: The average load of a multi-constraint query grows sub-linearly with the system size.

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 7 shows that queries are resolved quickly regardless of k. Average query latency is determined by the slowest node in each ring (subject to the maximum latency bound) and the hop count, both of which increases only marginally as k increases from four to sixteen.

The β parameter captures the tradeoff between query latency and accuracy as shown in Figure 8. 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, and also Thm. 4.3 and Thm. 4.4a). 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 9 plots the median error and average query latency for $k \sim \log N$. 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 plots the total bandwidth required throughout the entire network to resolve a query, and shows that it grows sub-linearly with system size, with 2000 nodes requiring a total of 2.6 KB per query.

A desirable property for load-balancing, and one of the assumptions in our theoretical analysis (see Thm. 4.4b on 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 1; in other words, it would indicate that the nodes within the radius r around A are selected evenly as neighbors. Figure 11 shows that Meridian is very evenly load-balanced, as more than 90% of the balls have an in-degree ratio less than two for balls of radius 20ms and 50ms.

A desirable property, and one of the assumptions in our theoretical analysis (see Thm. 4.2) is that our ring members are well distributed due to our multi-resolution ring structure and our hypervolume ring membership replacement scheme. To determine their actual effectiveness, we evaluate 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. The CDF in Figure 12 indicates that for $\beta = 0.5$, 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 gives a good indication that multi-resolution rings and hypervolume ring membership replacement protocol are doing a good job in distributing the ring nodes in the latency space. The hypervolume ring membership protocol also provides significantly more consistent results than a random replacement protocol, as the standard deviation of relative error is 38ms using hypervolume replacement, but is 151ms when using random replacement.

We evaluate how Meridian performs in central leader election by measuring its relative error as a function of group size. Figure 13 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 Meridian's task of routing the query to a suitable node.

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 the constraint to each target node is drawn from a uniform distribution 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 14 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%.

As in closest node discovery, the number of nodes per ring k has the largest influence on the performance of the multi-constraint protocol. Figure 15 shows the failure rate decreases as the number of nodes per ring increases. Surprisingly, it also shows a decrease in average query latency as the number of nodes per ring increases. This is due to the reduction in the number of hops needed before a constraint satisfying node is found, as a search can end early by finding a satisfactory node. Figure 16 shows that varying β in the multi-constraint protocol has similar but less pronounced effects as in the closest node discovery protocol. An increase in β decreases the failure percentage and increases the average latency of a multi-constraint query.

The scalability properties of the multi-constraint system are very similar to the scalability of closest node discovery. Figure 17 shows that the failure rate and the average query latency are independent of system size, even when the number of nodes per ring $k \sim \log N$ (note that setting k to a constant would favor the runs with small N). Figure 18 shows that the average load per multi-constraint query grows sub-linearly. 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 the closest node discovery protocol on PlanetLab. The implementation is small, compact and straightforward; it consists of approximately 2500 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

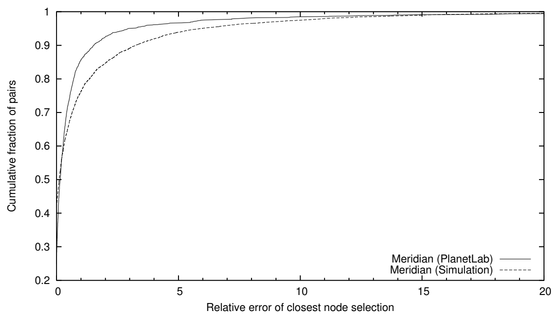

Figure 19: The Meridian (PlanetLab) curve shows the relative error of a deployment of Meridian over 166 PlanetLab nodes. We compared it against a similarly configured simulation to ensure the validity of our simulation results.

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.

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 $k = 8$, $s = 2$ms, $\beta = 0.5$, and $\alpha = 1$. Overall, median error in Meridian is 1.844ms, and the relative error CDF in Figure 19 shows that it performs better than simulation results from a similarly configured system.

6 RELATED WORK

Meridian is a general node proximity framework that we have applied to the server selection. We separate the server selection techniques into those that require network embedding 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 the 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 [40] is the pioneer in network embedding systems. It uses a fixed set of landmarks that determines the coordinates of a node by its distance to the landmarks. ICS [36] and Virtual Landmarks [52] both aim to reduce the computational cost of GNP by replacing the embedding with ones that are computationally cheaper, at the cost of losing accuracy. Meridian uses the same low-cost embedding as Virtual Landmarks, but employs the resulting coordinates only for selecting diverse ring members, not for resolving queries. To address the issue of single point of failure due to fixed landmarks, Lighthouse [41] uses multiple local coordinate system that are joined together through a transition matrix to form a global coordinate system. PIC [9] and PCoord [35] only require fixed landmarks 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 [39] is similar to PIC and PCoord but further imposes a hierarchy of servers to ensure consistency of the coordinates across all the nodes. Vivaldi [12] is based on a simulation of springs, where the position of the nodes that minimizes the potential energy of the spring also minimizes the embedding error. BBS [48] performs a similar simulation to calculate coordinates, simulating an explosion of particles under a force field.

IDMaps [16], like network embedding systems, 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 [15] has also examined how to delegate probing to specialized nodes in the network.

There has also been theoretical work [28, 49] on explaining the empirical success of network embeddings and IDMaps-style approaches.

Server Selection: Our closest node discovery protocol draws its inspiration from DHTs such as Chord [50], Pastry [45] and Tapestry [54], but these DHTs solve a different problem, namely routing. Proximity based neighbor selection [6, 5] performs a similar search using the node entries in the route table of a structured P2P system. This technique relies on the routing table levels to loosely characterize peer nodes by latency, but does not directly organize nodes based on their latency and incurs the overhead associated with structured P2P systems. The time and space complexity of two similar techniques are discussed in [24] and [26], but these techniques do not provide a general framework, and instead focus exclusively on finding the nearest neighbor. Moreover, their results appear to apply only to Internet latencies modeled by growth-constrained metrics, whereas our framework extends to a more general model (see Section 4). Also, without an evaluation on a large scale data set collected from live Internet nodes, their practicality can not be confirmed.

A closest node discovery technique described as *beaconing* is introduced in [29], where k landmarks are placed, each keeping track of its latency to the nodes in the system. A node finds the closest node by querying all k landmarks for nodes that are the same distance $\pm\delta$ away from the landmarks, and choosing the closest node from that set. The accuracy of the system depends heavily on the assumption that triangle inequality holds on the majority of routes, and the choice of an appropriate δ is not obvious without prior knowledge of the node distribution.

Another landmark based technique for closest node discovery is described in [44], where each node determines its bin number via measurements to well known landmarks. A node wishing to find its closest node determines its own bin number, queries a modified DNS server for other nodes in the same bin, or the nearest bin if no other nodes belong in the same bin, and chooses a random node from the retrieved set of servers.

Several different proactive techniques to locate the closest replica to the client are evaluated in [19]. These techniques offer different methods to construct a connectivity graph by means of polling the routing table of connecting hops, explicitly sending routing probes, or limited probing with triangulation. The study assumes the network conditions and topology remain relatively static, and does not directly address scalability.

Dynamic server selection was found in [3] 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 [4] 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

Network positioning based node selection 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 such as CAN [43] and P-Trees [10].

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

The authors are grateful to Frank Dabek, Russ Cox, Frans Kaashoek, Robert Morris, Eugene Ng and Hui Zhang for sharing the Vivaldi and GNP software and data.

References

[1] D. Andersen, H. Balakrishnan, M. Kaashoek, and R. Morris. Resilient Overlay Networks. In *SOSP 2001*.

[2] 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 *NSDI 2004*.

[3] R. Carter and M. Crovella. Server Selection Using Dynamic Path Characterization in Wide-Area Networks. In *INFOCOM 1997*.

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

[5] M. Castro, P. Druschel, Y. Hu, and A. Rowstron. Exploiting network proximity in peer-to-peer overlay networks. In *Tech Report MSR-TR-2003-82*, Microsoft Research, 2002.

[6] M. Castro, P. Druschel, Y. Hu, and A. Rowstron. Proximity neighbor selection in tree-based structured peer-to-peer overlays. In *Tech Report MSR-TR-2003-52*, MSR, 2003.

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

[8] Y. Chu, S. Rao, and H. Zhang. A Case for End System Multicast. In *SIGMETRICS 2000*.

[9] M. Costa, M. Castro, A. Rowstron, and P. Key. PIC: Practical Internet Coordinates for Distance Estimation. In *ICDCS 2004*.

[10] A. Crainiceanu, P. Linga, J. Gehrke, and J. Shanmugasundaram. Querying Peer-to-Peer Networks Using P-Trees. In *WebDB 2004*.

[11] W. Cui, I. Stoica, and R. Katz. Backup Path Allocation Based On A Correlated Link Failure Probability Model In Overlay Networks. In *ICNP 2002*.

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

[13] F. Dabek, M. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *SOSP 2001*.

[14] 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 *PODC 1987*.

[15] 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 1998*.

[16] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. IDMaps: A global Internet host distance estimation service. *IEEE/ACM Transactions on Networking*, 9:525–540, October 2001.

[17] K. Gummadi, S. Saroiu, and S. Gribble. King: Estimating Latency between Arbitrary Internet End Hosts. In *IMW 2002*.

[18] A. Gupta, R. Krauthgamer, and J. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *FOCS 2003*.

[19] J. Guyton and M. Schwartz. Locating Nearby Copies of Replicated Internet Servers. In *SIGCOMM 1995*.

[20] J. Heinonen. Lectures on analysis on metric spaces. Springer Verlag, Universitext 2001.

[21] K. Hildrum, R. Krauthgamer, and J. Kubiatowicz. Object location in realistic network. In *SPAA 2004*.

[22] K. Hildrum, J. Kubiatowicz, S. Ma, and S. Rao. A note on finding the nearest neighbor in growth-restricted metrics. In *SODA 2004*.

[23] 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.

[24] K. Hildrum, J. Kubiatowicz, S. Rao, and B. Zhao. Distributed Object Location in a Dynamic Network. In *SPAA 2002*.

[25] K. Johnson, J. Carr, M. Day, and M. Kaashoek. The measured performance of content distribution networks. In *WCW 2000*.

[26] D. Karger and M. Ruhl. Finding Nearest Neighbors in Growth-restricted Metrics. In *STOC 2002*.

[27] D. Kempe, J. Kleinberg, and A. Demers. Spatial Gossip and Resource Location Protocols. In *STOC 2001*.

[28] J. Kleinberg, A. Slivkins, and T. Wexler. Triangulation and embedding using small sets of beacons. In *FOCS 2004*.

[29] C. Kommareddy, N. Shankar, and B. Bhattacharjee. Finding Close Friends on the Internet. In *ICNP 2001*.

[30] R. Krauthgamer and J. Lee. Navigating nets: simple algorithms for proximity search. In *SODA 2004*.

[31] R. Krauthgamer and J. Lee. The black-box complexity of nearest neighbor search. In *ICALP 2004*.

[32] R. Krauthgamer and J. Lee. The intrinsic dimensionality of graphs. In *STOC 2003*.

[33] R. Krauthgamer, J. Lee, M. Mendel, and A. Naor. Measured descent: a new embedding method for finite metrics. In *FOCS 2004*.

[34] R. Lawrence. Running Massively Multiplayer Games as a Business. In *Keynote: NSDI 2004*.

[35] L. Lehman and S. Lerman. PCoord: Network Position Estimation Using Peer-to-Peer Measurements. In *NCA 2004*.

[36] H. Lim, J. Hou, and C. Choi. Constructing Internet Coordinate System Based on Delay Measurement. In *IMC 2003*.

[37] P. Maniatis, M. Roussopoulos, T. Giuli, D. Rosenthal, M. Baker, and Y. Muliadi. Preserving peer replicas by rate-limited sampled voting. In *SOSP 2003*.

[38] R. Motwani and P. Raghavan. Randomized algorithms. Cambridge University Press, 1995.

[39] T. Ng and H. Zhang. A Network Positioning System for the Internet. In *USENIX 2004*.

[40] T. Ng and H. Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *INFOCOM 2002*.

[41] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti. Lighthouses for Scalable Distributed Location. In *IPTPS 2003*.

[42] C. Plaxton, R. Rajaraman, and A. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *SPAA 1997*.

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

[44] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-Aware Overlay Construction and Server Selection. In *INFOCOM 2002*.

[45] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Middleware 2001*.

[46] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *SOSP 2001*.

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

[48] Y. Shavitt and T. Tankel. Big-Bang Simulation for Embedding Network Distances in Euclidean Space. In *INFOCOM 2003*.

[49] A. Slivkins. Distributed approaches to triangulation and embedding. In *SODA 2005*.

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

[51] K. Talwar. Bypassing the embedding: approximation schemes and compact representations for growth restricted metrics. In *STOC 2004*.

[52] L. Tang and M. Crovella. Virtual Landmarks for the Internet. In *IMC 2003*.

[53] H. Weatherspoon, T. Moscovitz, and J. Kubiatowicz. Introspective Failure Analysis: Avoiding Correlated Failures in Peer-to-Peer Systems. In *RPPDS 2002*.

[54] 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.

A Proofs

Here we provide proofs for the results in Section 4. First we address the quality of the rings, then approximate nearest neighbors, then exact nearest neighbors, and conclude with load-balancing; the proof on load-balancing is significantly more complicated than the other proofs. To make this write-up self-contained, we include a subsection on Chernoff Bounds that are used throughout the proofs.

For simplicity we redefine the KR-dimension as the smallest α such that for any $\beta > 1$ the cardinality of any ball $B_u(r)$ is at most β^α times larger that of $B_u(r/\beta)$. It is easy to check that this definition coincides with the old one for any $\beta = 2^i$, $i \geq 1$. We redefine the doubling dimension similarly.

If w is a neighbor of the current node u, and q is the target, then call w a *progress-β* neighbor if $d_{uq}/d_{wq} \geq \beta$.

A.1 Quality of the rings

We start with two proofs which show that even with small k it is possible to make the rings ϵ-nice.

Proof of Thm. 4.1: Fix two Meridian nodes uv 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 $\gamma = 4 + 3/\epsilon$, so $|B_{ui}| \leq \gamma^\alpha |B_v(r)|$. Therefore by Chernoff Bounds (Claim A.10) some node from S_{ui} lands in $B_v(r)$ with failure probability at most δ/N^2. □

Proof of Thm. 4.5: Fix two Meridian nodes uv and let $r = \epsilon d_{uv}$. Pick the smallest i such that $d_{uv} + r \leq 2^i$. Then $r > 2^i\gamma$, where $\gamma = 2(1 + \frac{1}{\epsilon})$. So applying the definition of a doubling measure $\log\gamma$ times we see that $\mu[B_{ui}]/\mu[B_v(r)] \leq 2^{O(\alpha \log \gamma)} = \gamma^{O(\alpha)}$.

The ring S_{ui} is distributed as in the following process: pick nodes from B_{ui} independently with probability $\mu(\cdot)/\mu(B_{ui})$, until we gather k distinct nodes. At each draw the probability of choosing a node from $B_v(r)$ is at least $\gamma^{-O(\alpha)}$. The claim follows from the Chernoff Bounds (Lemma A.9), exactly as in Claim A.10. □

A.2 Approximate nearest neighbor

In this subsection we prove Thm. 4.2.

The search algorithm used by Meridian (denoted by $\mathcal{A}(\beta)$ in Section 4) looks only at three rings at a given node. For Thm. 4.2 we'll need a generalization $\mathcal{A}(\beta, l)$, which looks at $l \geq 3$ rings. Specifically, if node u receives a query for target q, it chooses $i = 2 + \lfloor \log d_{uq} \rfloor$, and finds a neighbor w in the l rings S_{uj}, $i - l \leq j < i$ that is closest to q. If w is a progress-β neighbor then the query is forwarded to w; else the search stops.

The following claim essentially shows that if we look at the ring of radius that is too small then we cannot make much progress towards the target node.

Claim A.1 *For any nodes (u, w, q), suppose $d_{uw} \leq 2^{i-j}$, $i = \lfloor \log d_{uq} \rfloor$. Then $d_{uq}/d_{wq} < 1 + 2^{2-j}$.*

Proof: Let $\gamma = 2^{1-j}$. Then $d_{uw} \leq \gamma 2^{i-1} < \gamma d_{uq}$, so $d_{wq} \geq d_{uq} - d_{uw} > (1-\gamma) d_{uq}$, and it follows that $d_{uq}/d_{wq} < 1/(1-\gamma) \leq 1 + 2\gamma$. □

By Claim A.1 in a given step we might not need to look at all l rings: we look at the rings S_{uj} in the order of decreasing j, and without loss of generality we consider the ring S_{uj}, $j \leq i - 4$ only if in the larger rings there was no node v such that $d_{uq}/d_{vq} \geq 1 + 2^{j-i+4}$. In particular, if for some l algorithm $\mathcal{A}(\beta, l)$ finds a progress-2 node then so does $\mathcal{A}(\beta, 3)$.

The following claim shows how our algorithm $\mathcal{A}$ zooms in on the target node. We'll use the function

$$f(\beta) = \beta(1+\epsilon)/(1-\beta\epsilon).$$

Note that for $\beta \in (1, \frac{1}{\epsilon})$ the function $f(\beta)$ is continuously increasing to infinity. Define $l(\beta) = 3$ if $\beta \geq 2$ and $l(\beta) = 3 + \lfloor \log 1/(2-\beta) \rfloor$ otherwise.

Claim A.2 *Assume the rings are ϵ-nice. Let q be the target node, let $v \in S_M$ be its nearest neighbor. Let u be any Meridian node, suppose $d_{uq}/d_{vq} = f(\beta)$ for some $\beta \in (1, \frac{1}{\epsilon})$, and fix $l \geq l(\beta)$. Then at u algorithm $\mathcal{A}(\beta, l)$ finds a progress-β neighbor of u.*

Proof: First we claim that such neighbor exists. Indeed, pick the smallest i such that $d_{uv}(1+\epsilon) \leq 2^i$. Since the rings are ϵ-nice, node u has a neighbor $w \in S_{ui}$ within distance ϵd_{uv} from node v. Then, letting $d = d_{vq}$,

$$\begin{aligned} d_{wq} &\leq d + d_{vw} \leq d + \epsilon d_{uv} \leq d + \epsilon(d + d_{uq}) \\ &= \epsilon d_{uq} + (1+\epsilon) d_{uq}/f(\beta) = d_{uq}/\beta, \end{aligned}$$

claim proved. It remains to show that w lies in one of the rings considered by the algorithm $\mathcal{A}(l)$, i.e. that if $d_{uq} \leq 2^j < 2d_{uq}$ then $j - l \leq i \leq j + 1$. Indeed, $i \leq j + 1$ follows since

$$\begin{aligned} d_{uv} &\leq d_{uq} + d \leq d_{uq}(1 + f(\beta)^{-1}) \\ &\leq d_{uq}(1 + f(1)^{-1}) \leq 2d_{uq}/(1+\epsilon), \\ 2^i &< 2d_{uv}(1+\epsilon) \leq 4d_{uq} \leq 2^{j+2}, \end{aligned}$$

and $j - l \leq i$ follows by Claim A.1. □

The next claim allows us to use Claim A.2 for small β; the proof is straightforward.

Claim A.3 *For any $\gamma \in (0, 1)$ we have*

$$f(1+\gamma)/(1+\gamma) \leq f(1+\gamma/2).$$

Moreover, $f(1+\epsilon^2/2) \leq 1 + 3\epsilon$.

Now we are ready to prove Thm. 4.2.

Proof of Thm. 4.2: Let q be the target of the nearest-neighbor query, and let d be the distance from q to its closest Meridian neighbor.

(a) We need to prove that algorithm $\mathcal{A}(2,3)$ finds a 3-approximate neighbor of q. By Claim A.2 while the query visits nodes u such that $d_{uq}/d \geq f(2)$, the algorithm finds a progress-2 neighbor of u and forwards the query to it. The distance to q goes down by a factor of at least 2 at each step, so after at most $\log(\Delta)$ steps the query should arrive at some node v such that $d_{vq}/d < f(2) \leq 3$. This proves part (a).

(b) We'll show that $\mathcal{A}(\beta, l)$ finds a $(1+3\epsilon)$-approximate neighbor of q, where $l = 3 + \lfloor \log 2/\epsilon^2 \rfloor$. By Claim A.2 while the query visits nodes u such that $d_{uq}/d \geq f(2)$, the distance to q goes down by a factor of at least 2 at each step. So after at most $\log(\Delta)$ steps the query should arrive at some node v such that $d_{vq}/d < f(2)$. Then by Claims A.2 and A.3, using induction on i we show that after $i < l$ more steps the query will arrive at node w such that $d_{wq}/d < f(1+2^{-i})$. In particular, by Claim A.3 we are done when $i = \log(\epsilon^2/2)$. This proves part (b).

(c) Note that if any neighbor of the current node u achieves progress less than $1+\gamma$, $\gamma \in (0, \frac{1}{2})$ then by Claim A.2 we have $d_{uq}/d \leq f(1+\gamma) \leq 1 + 3\epsilon + 2\gamma$. □

A.3 Exact nearest neighbor

Here we prove Thms. 4.3 and 4.4a on finding exact nearest neighbors. In both theorems the progress is at least 2 at every 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.

Proof of Thm. 4.3: Let $k = 2.2 \cdot 10^{\alpha} \ln(N|Q|/\delta)$. Let $q \in Q$ be the target, and let $v \in S_M$ be its exact nearest neighbor. Fix some Meridian node u, let $d = d_{uq}$ and choose i such that $2d \leq 2^i < 4d$.

We claim that either $v \in S_{ui}$, or with failure probability at most $\delta/N|Q|\log(\Delta)$ the ring S_{ui} contains some $w \in B_v(d/2)$. Indeed, $B_{ui} \subset B_q(5d)$, so

$$|B_{ui}| < |B_q(5d)| \leq 10^{\alpha} |B_q(d/2)|,$$

so if $|B_{ui}| \geq k$ then the claim follows from Claim A.10; the constant 2.2 in front of k works numerically as long as e.g. $n|Q| > 55^2$ and $\delta < e^{-2}$, which is quite reasonable. Finally, if $|B_{ui}| \leq k$ then every node in B_{ui} is in ring S_{ui}, including v, claim proved.

So the progress is at least 2 at every step except maybe the last one, with failure probability at most $\delta/N|Q|$. The Union Bound over all $N|V|$ possible uq pairs gives the total failure probability δ. □

Thm. 4.4a is proved using the same idea, except we need to address the fact that Meridian nodes themselves are chosen at random from Q. Let $Q_u(r)$ denote the closed ball in Q of radius r around node u, i.e. the set of all nodes in Q within distance r from u.

Proof of Thm. 4.4a: Denote $Q_{ui} = Q_u(2^i)$ and let $k = 8 \cdot 10^{\alpha} \ln(2N|Q|/\delta)$. Let q be the target and let $v \in S_M$ be its exact nearest neighbor. Fix some Meridian node u, let $d = d_{uq}$ and $B = B_q(d/2)$, and choose i such that $2d \leq 2^i < 4d$.

Note that we can view the process of selecting S_M from Q as follows: choose the cardinality y for B_{ui} from the appropriate distribution, then choose, independently and uniformly at random, y nodes from Q_{ui}, and $n-y$ nodes from $Q \setminus Q_{ui}$.

We claim that with failure probability at most $\delta' = \delta/N|Q|$ either $v \in S_{ui}$, or the ring S_{ui} contains some $w \in B$. Indeed, if the cardinality of B_{ui} is at most k, then all of B_{ui} lies in the ring S_{ui}, including v. Now assume the cardinality of B_{ui} is some fixed number $x > k$. Since $Q_{ui} \subset Q_q(5d)$, it follows that

$$\frac{x}{E(|B|)} = \frac{|Q_{ui}|}{|Q_u(d/2)|} \leq \frac{|Q_u(5d)|}{|Q_u(d/2)|} \leq 10^{\alpha},$$

so by Claim A.11a with failure probability at most $\delta'/2$ the cardinality of B is at least half the expectation, so that by Claim A.10 with failure probability at most $\delta'/2$ some node in ring S_{ui} lands in B. Claim proved.

Therefore the progress is at least 2 at every step except maybe the last one, with failure probability at most δ'. The Union Bound over all $N|V|$ possible uq pairs gives the total failure probability δ. □

A.4 Load-balancing: Thm. 4.4b

In this subsection we'll prove Thm. 4.4b, which is about load-balancing. A large part of the proof is the setup: it is non-trivial to restate the algorithm and define the random variables so that the forth-coming Chernoff Bounds-based argument works through. For convenience, for any $x > 0$ denote $[x] = \{0, 1 \ldots \lceil x \rceil\}$.

For technical reason we'll need a slightly modified search algorithm. On every step in algorithms $\mathcal{A}(\cdot)$ and $\mathcal{A}'(\cdot)$ we look at a subset S of neighbors, and either the search stops, or the query is forwarded the node $w \in S$ that is closest to the target. Here is the modification: if w is a progress-2 node, then instead of forwarding to w the algorithm can forward the query to an arbitrary progress-2 node in S. It is easy to check that all our results for $\mathcal{A}(\cdot)$ and $\mathcal{A}'(\cdot)$ carry over to this modification. For Thm. 4.4b we'll need a specific version of $\mathcal{A}'(2)$ which can be seen as a rule to select between different progress-2 nodes.

As compared to (the proof of) part (a), we increase the ring cardinalities by a factor of $O(\log N)(\log \Delta)$. This is essentially because we need more randomness, so that in the proof we could use Chernoff Bounds more efficiently.

While it might be possible to prove the theorem without this blow-up, it seems to lead to mathematical difficulties that are way beyond the scope of this paper.

Recall that the S_{ui} is well-formed if it is distributed as a random k-node subset of B_{ui}. Here for technical convenience we'll use a slightly different definition: say S_{ui} is *well-formed* if it is distributed as a set of k nodes drawn independently uniformly at random from B_{ui}. The difference is that the new definition allows repetitions; note that all previous results work under either definition.

Let's define our version of $\mathcal{A}'(2)$, which we denote $\mathcal{A}$. Say each ring S_{ui} consists of k *slots* $\zeta_{ui}(j)$ which can be seen as independent random variables distributed uniformly at random in B_{ui}. Since the rings are independent,

$$\{\zeta_{ui}(j) : u \in S_M, i \in [\log\Delta], j \in [k]\}$$

is a family of independent random variables.

Let $L = 6\ln(N\log(\Delta)/\delta)$. For every pair ui, partition the slots $\zeta_{ui}(\cdot)$ into $L\log(\Delta)$ equal-size collections $\mathcal{C}_{ui}(j,l)$, where $j \in [\log\Delta]$ and $l \in [L]$; formally, each such collection is a set of indices j' into $\zeta_{ui}(j')$. Let

$$S_{ui}(j,l) = \{\zeta_{ui}(j') : j' \in \mathcal{C}_{ui}(j,l)\} \subset B_{ui}$$

be the set of *values* of slots in $\mathcal{C}_{ui}(j,l)$. Obviously, the union of all sets $S_{ui}(\cdot,\cdot)$ is S_{ui}.

Say a *j-step query* is a query on the j-th step of the algorithm. When node u receives a j-step query to target q, it chooses $l \in [L]$ in a round-robin fashion (the round-robin is separate for each uj pair) and lets algorithm $\mathcal{A}(1)$ handle this query using only the neighbors in $S_{ui}(j,l)$, for the corresponding i. Specifically, $\mathcal{A}(1)$ sets $i = 1 + \lfloor\log d_{uq}\rfloor$, asks every node in $S_{ui}(j,l)$ to measure the distance to q, and forwards the query to the closest one.

We make each collection $\mathcal{C}_{ui}(j,l)$ have size $k' = 8 \cdot 10^{\alpha}\ln(2K/\delta)$, where $K = N|Q|L\log(\Delta)$. Then using the argument from part (a) we can show that for given (u,q,j,l) either the corresponding $S_{ui}(j,l)$ contains a progress-2 node or it contains a nearest neighbor of q, with failure probability at most δ/K. The Union Bound over all K possible (u,q,j,l) tuples shows that our algorithm is Q-exact with failure probability δ.

Note that algorithm $\mathcal{A}$ can be seen as $\mathcal{A}'(l)$ with a rule to select between different progress-2 nodes: namely, choose a progress-2 node from the corresponding $S_{ui}(j,l)$ if such node exists, else proceed with $\mathcal{A}'(l)$. Obviously, in under assumptions of Thm. 4.4a with failure probability δ this scheme will behave exactly as $\mathcal{A}$.

We consider a scenario where many independent random choices are made. Specifically we choose

- N-node subset S_M of Q,
- subsets $S_{ui}(j,l) \subset B_{ui}$ for each tuple (u,i,j,l),
- target t_u for each node u.

For a collection of independent random choices, without loss of generality we can assume that a given choice happens any time before its result is actually used. In particular, we will assume that at first, S_M and t_u's are chosen. Then the time proceeds in $\log(\Delta)$ epochs; in a given epoch j, all subsets $S_{ui}(j,\cdot)$ are chosen, then all queries are advanced for one step.

Let's analyze the choice of S_M and the queries. Let T be the set of all N queries. For $q \in T$, let $t(q)$ be the corresponding target. Let $T_v(r)$ be the set of queries $q \in T$ such that $t(q)$ is within distance r from v. Let $t(S)$ be the set of all targets in the set S of queries. Let $\gamma = N/|Q|$. By Claim A.11 $|B_u(r)|$ and $|T_u(r)|$ are close to its expectation:

Claim A.4 *With failure probability at most δ, for any $u \in S_M \cup t(T)$ and radius r the following holds:*

(*) if $z = \gamma|Q_u(r)| \geq k_0$ then $|B_u(r)|$ and $|T_u(r)|$ are within a factor of 2 from z, else they are at most $2k_0$, where $k_0 = O(\log(n/\delta))$.

This completes the setup; now, finally, we can argue about load-balancing. We need some more preliminaries. Recall that all queries are handled separately, even if a given node simultaneously receives multiple queries for the same target. When node u handles a j-step query and in the process measures distance to its neighbor v, we say that v receives a *j-step request* from u. Let's define several families of random variables:

- $X_{uv}(j,l)$ is the number of j-step queries forwarded from u to v, and handled at u using a set $S_{ui}(j,l)$, for some i.
- X_u^j is the number of all j-step queries forwarded to node u; set $X_u^0 = 1$. Then $X_{uv}(j,l) \leq X_u^j/L$.
- $Y_{uv}(j,l)$ is the number of j-step requests received by v from u, and handled at u using a set $S_{ui}(j,l)$, for some i.
- Y_u^j is the number of all j-step requests received by node u. Clearly, $Y_{uv}(j,l) \leq X_u^{j-1}/L$.

For every j-step query received, a given node sends some constant number c of packets to each of the k' neighbors in the corresponding set $S_{ui}(j,l)$. Therefore a given node u sends $ck'\sum_j X_u^j$ packets total, and receives $c\sum_j Y_u^j$ packets total. Since a single query involves exchanging at most $ck'\log(\Delta)$ packets, algorithm $\mathcal{A}$ is (β,Q)-balanced if and only if $\sum_j(k'X_u^j + Y_u^j) \leq 2\beta k'\log(\Delta)$ for every node u.

Say property $P(j)$ holds if for each node v it is the case that $X_v^j \leq \beta$ and $Y_v^j/k' \leq \beta$. We need to prove that with high probability $P(j)$ holds for all j. It suffices to prove the following claim:

Claim A.5 *If $P(j-1)$ then $P(j)$, with failure probability at most $\delta/\log(\Delta)$.*

Then we can take the Union Bound over all $\log \Delta$ steps to achieve the desired failure probability δ.

Let's prove Claim A.5. Suppose all queries have completed $j-1$ steps and are assigned to the respective sets $S_{ui}(j,l)$. Now the only remaining source of randomness before the j-th step is the choice of these sets. In particular, each random variable $X_{uv}(j,l)$ depends only on one set $S_{ui}(j,l)$, and so does $Y_{uv}(j,l)$. Since these sets are chosen independently, for any fixed v variables

$$\{X_{uv}(j,l) : u \in S_M, l \in [L]\}$$

are independent, and so are

$$\{Y_{uv}(j,l) : u \in S_M, l \in [L]\}.$$

First we claim that $P(j)$ holds in expectation:

Claim A.6 *For every Meridian node v and every step j (a) $E(X_v^j) \leq \beta/2$ and (b) $E(Y_v^j/k') \leq \beta/2$.*

Suppose property $P(j-1)$ holds. Let's bound the load on some fixed node v. Note that

$$X_v^j = \sum_{\text{all pairs } (u,l)} X_{uv}(j-1,l)$$

is a sum of independent random variables, each in $[0,y]$ for $y = \beta/L$. Applying Claim A.9b with $\mu = \beta/2$, we see that

$$\Pr[X_v^j > \beta] \leq (e/4)^{L/2} \leq \delta/2N\log(\Delta).$$

Similarly, $Y_v^j = \sum_{(u,l)} Y_{uv}(j,l)$ is a sum of independent random variables, each in $[0,y]$, so by Claim A.9b we can upper-bound $\Pr[Y_v^j/k' > \beta]$. By the Union Bound property $P(j)$ holds with the total failure probability at most δ. This completes the proof of Claim A.5.

It remains to prove Claim A.6. Let S_0 be the set of queries $q \in T$ such that v is a nearest neighbor of the target $t(q)$.

Claim A.7 $|S_0| \leq O(2^\alpha)\log(N/\delta)$.
Proof: Choose target $t \in t(S_0)$ such that d_{vt} is maximal. Let $d = d_{vt}$. Then $B_t(d/\tau) \in \{q\}$ for any $\tau > 1$, so by Claim A.4 $|Q_t(d/\tau)| \leq O(\log(n/\delta))$. Note that $S_0 \subset B_t(2d) \subset Q_t(2d)$ and

$$|Q_t(2d)| \leq (2\tau)^\alpha |Q_t(d/\tau)| \leq (2\tau)^\alpha O(\log(n/\delta)).$$

Claim follows if we take small enough $\tau > 1$. □

Let r_0 be the smallest r such that $B_v(r)$ has cardinality at least twice the k_0 from Claim A.4. Let $R_i = T_v(r_0\, 2^i)$. Let $S \subset T$ be the set of queries that get forwarded to v on step j; recall that $X_v^j = |S|$.

Claim A.8 *For any query $q \in T \setminus (S_0 \cup R_0)$ and $t = t(q)$ we have $\Pr[q \in S] \leq O(2^\alpha)/|B_v(d_{vt})|$.*
Proof: Let $d = d_{vt}$ and suppose query q is currently at node u. Since $q \notin S_0$ this query gets forwarded to some node $w \in B_q(d_{ut}/2)$, so if $d > d_{ut}/2$ then clearly $q \notin S$. Assume $d \leq d_{ut}/2$. Since $B_v(d) \subset B_t(2d)$, by Claim A.4 we have

$$\begin{aligned} |B_v(d)| &\leq |B_t(2d)| \leq 2\gamma |Q_t(2d)| \leq 2\gamma\, 2^\alpha |Q_t(d)| \\ &\leq 4\, 2^\alpha |B_t(d)|, \\ \Pr[q \in S] &= 1/|B_t(d_{ut}/2)| \leq 1/|B_t(d)|, \end{aligned}$$

which is at most $4\, 2^\alpha/|B_v(d)|$, as required. □

Now for $R = R_{i+1} \setminus (R_i \cup S_0)$ and $r = r_0\, 2^i$

$$\begin{aligned} \gamma_i &:= E|S \cap R| \leq |R_{i+1}| \Pr[q \in S : q \in R] \\ &\leq O(2^i)|R_{i+1}|/|R_i| \leq O(4^\alpha), \\ E|X_v^j| &= E|S| \leq |S_0| + |R_0| + \sum \gamma_i \\ &\leq O(2^\alpha)\log(n/\delta) + O(4^\alpha)\log(\Delta) \\ &\leq O(4^\alpha)\log(n\Delta/\delta) \leq \beta/2. \end{aligned}$$

This completes the proof of Claim A.6a. For Claim A.6b, let S be the set of queries that cause a j-step request to v. Suppose a j-step query q is at node u; let $t = t(q)$ and $d = d_{ut}$. Node v receives a j-step request due to t only if $d_{uv} \leq 2d$, so let's assume it is the case. Then $d_{vt} \leq d + d_{uv} \leq 3d$, so

$$\begin{aligned} B_u(d_{vt}) &\subset B_u(d_{uv} + d_{vt}) \subset B_u(5d) \\ |B_u(d_{vt})| &\leq |B_u(5d)| \leq 4\,(2.5)^\alpha\, |B_u(2d)| \\ \Pr[v \in S] &\leq 1/|B_u(2d)| \leq 4\,(2.5)^\alpha\, |B_u(d_{vt})| \end{aligned}$$

as long as $|B_u(d_{vt})|$ is at least twice as large as the k_0 from Claim A.4. The rest of the proof of Claim A.6b is similar to that of Claim A.6a. This completes the proof of Claim A.6 and Thm. 4.4b.

A.5 Chernoff Bounds

Essentially, Chernoff Bounds say that with high probability the sum of many bounded independent random variables is close to its expectation. Here for the sake of completeness we write out the standard Chernoff Bounds and some easy applications thereof that we use in the above proofs.

Lemma A.9 (Chernoff Bounds [38]) *Let X be the sum of independent random variables $X_i \in [0,y]$, for some $y > 0$. Let $\epsilon \in (0,1)$ and $\beta \geq 1$. Then:*

(a) $\Pr[X < (1-\epsilon)\mu] \leq e^{-\epsilon^2\mu/2y}$, *for any $\mu \leq E(X)$.*

(b) $\Pr[X > \beta\mu] \leq \left[\frac{1}{e}(\frac{e}{\beta})^\beta\right]^{\mu/y}$, *for any $\mu \geq E(X)$.*

Claim A.10 *Suppose ring S_{ui} is well-formed and has cardinality k. Fix a subset $S \subset B_{ui}$ and let $\mu = k|S|/|B_{ui}|$. Then with failure probability at most $e^{-(1-1/\mu)^2\mu/2}$ some node from S_{ui} lands in S.*

Proof: Denote the desired event by A. The distribution of S_{ui} is that of the following process P: pick nodes from B_{ui} independently and uniformly at random, until we gather k_{ui} distinct nodes. For simplicity consider a slightly modified process P': pick k_{ui} nodes from B_{ui} independently and uniformly at random, possibly with repetitions. Obviously, P' is doing exactly the same as P, except P might stop later and, accordingly, choose some more nodes. Therefore $\Pr_P[A] \geq \Pr_{P'}[A]$.

Let's analyze process P'. Let X_j be a 0-1 random variable that is equal to 1 if and only if the j-th chosen node lands in $B_v(r)$. Then $\Pr[X_j = 1] = |S|/|B_{ui}|$, so $\mu = E(\sum X_j)$. The claim follows from Lemma A.9a with $y = 1$ and $1 - \epsilon = 1/\mu$. □

Claim A.11 *Consider two sets $S' \subset S$ and suppose n nodes are chosen independently and uniformly at random from S; say X of these n nodes land in S'. Let $\lambda = n|S'|/|S|$. Then*

(a) $\Pr[X < \lambda/2] \leq e^{-\lambda/8}$,

(b) $\Pr[X > k] \leq e^{-k/16}$ *for any $k \geq 2\lambda$,*

(c) $\Pr[X > 2\lambda] \leq (e/4)^{\lambda}$.

Proof: Let X_j be a 0-1 random variable that is equal to 1 if and only if the j-th chosen node lands in S'. Then $\Pr[X_j = 1] = |S'|/|S|$, so $X = \sum_{j=1}^{n} X_j$ and $\mu = E(X)$.

For part (a), use Lemma A.9a with $y = 1$ and $\epsilon = 1/2$. Parts (bc) follow from Lemma A.9b with $y = 1$ and $\beta = 2$; let $\mu = k/2$ in part (b), and let $\mu = \lambda$ in part (c). □