A Framework for Network Location-Aware Node Selection

BERNARD WONG, ALEKSANDRS SLIVKINS and EMİN GÜN SİRER
Cornell University

We introduce a lightweight, scalable and accurate framework for performing node selection based on network location. The framework, called Meridian, consists of an overlay network structured around multi-resolution rings, gossip protocols for ring maintenance, and query routing with direct measurements to satisfy user specified latency constraints. We show how this framework can be used to address three commonly encountered problems, namely, closest node discovery, central leader election, and locating nodes that satisfy target latency constraints in large-scale distributed systems without having to compute absolute coordinates. We also present the Meridian Query Language, a domain specific language for users to construct custom node selection queries based on their specific network location requirements. To facilitate adoption of Meridian, we have deployed a service called ClosestNode.com that provides a DNS to Meridian gateway for oblivious clients to initiate Meridian lookups. We show analytically that the framework is scalable with logarithmic convergence when Internet latencies are modeled as a growth-constrained metric, a low-dimensional Euclidean metric, or a metric of low doubling dimension. Large scale simulations, based on latency measurements from 6.25 million node-pairs as well as an implementation deployed on PlanetLab show that the framework is accurate and effective.

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

General Terms: Algorithms, Design, Measurement, Performance

Additional Key Words and Phrases: Node selection, Network locality, Nearest neighbor

1. INTRODUCTION

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

A general technique for finding nodes that optimize a given network metric is to perform a *network embedding*, that is, to map high-dimensional network measurements into a location in a smaller Euclidean space. For instance, recent work in network positioning [Ng and Zhang 2002; 2004; Dabek et al. 2004; Lim et al. 2003;

Tang and Crovella 2003; Shavitt and Tankel 2003; Pias et al. 2003; Costa et al. 2004; Lehman and Lerman 2004] uses large vectors of node-to-node latency measurements on the Internet to determine a corresponding single point in a d-dimensional space for each node. The resulting embedded address, a *virtual coordinate*, can be used to select nodes.

While the network embedding approach is applicable for a wide range of applications, it is neither accurate nor complete. The embedding process typically introduces significant errors. Selection of parameters, such as the constant d, the set of measurements taken to perform the embedding, the landmarks used for measurement, and the timing interval in which measurements are taken, is nontrivial and has a significant impact on the accuracy of the approach. Further, coordinates need to be recomputed as network latencies fluctuate. In addition, complex mechanisms besides virtual coordinates are required to support large-scale applications. Simple schemes, such as centralized servers that retain $O(N)$ state or naive algorithms with $O(N)$ running time, are unsuitable for large-scale networks. Peer-to-peer substrates that can naturally work with Euclidean coordinates and support range queries, such as CAN [Ratnasamy et al. 2001], Mercury [Bharambe et al. 2004] and P-Trees [Crainiceanu et al. 2004], can reduce the state requirements per node; however, these systems introduce substantial complexity and bandwidth overhead in addition to the overhead of network embedding. And our simulation results show that, even with a P2P substrate that always finds the best node based on virtual coordinates, the embedding error leads to a suboptimal choice.

This paper introduces a lightweight, scalable and accurate framework, called Meridian, for performing node selection based on network location [1]. Meridian forms a loosely-structured overlay network, uses direct latency measurements instead of latency estimates from virtual coordinates, and can solve spatial queries without an absolute coordinate space.

Each Meridian node keeps track of $O(\log N)$ peers and organizes them into concentric rings of exponentially increasing radii. This structure makes a node an expert in its own region of space, and provides it with sufficient contacts in far-away regions where it is not an authority. A query is matched against the relevant nodes in these rings, and optionally forwarded to a subset of the node's peers. Intuitively, the forwarding "zooms in" towards the solution space, handing off the query to a node that has more information to solve the problem due to the structure of its peer set. A scalable gossip protocol is used to notify other nodes of membership in the system. A node selection algorithm provides diverse ring membership to maximize the marginal utility provided by each ring member. Meridian avoids incurring embedding errors by making no attempt to reconcile the latencies seen at participating nodes into a globally consistent coordinate space. Directly evaluating queries against relevant peers in each ring avoids errors stemming from out of date coordinates. Meridian provides a general framework applicable for a wide range of network location problems. In this paper, we focus on three network location

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

problems that are commonly encountered in distributed systems, and describe how the lightweight Meridian framework can be used to address them.

The first network location problem that we examine is that of discovering the closest node to a targeted reference point. This is a pervasive problem; content distribution networks (CDNs) [Johnson et al. 2000], large-scale multiplayer games [Lawrence 2004], and peer-to-peer overlays [Hildrum et al. 2002; Karger and Ruhl 2002; Castro et al. 2002; 2003], among others, can significantly reduce response time and network load by selecting nodes close to targets. For instance, a geographically distributed peer-to-peer web crawler can reduce crawl time and minimize network load by delegating the crawl to the closest node to each target web server. Similarly, CDNs can reduce download time by assigning clients to nearby servers, and multiplayer games can improve gameplay by reducing server latency.

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

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

In addition to these commonly encountered problems, there are many other network location problems that are relevant to specific applications, or other solution strategies required for specific operating environments. To support these applications, we provide a language called the Meridian Query Language (MQL) for expressing application specific algorithms as queries, and a runtime for safe, online evaluation of these queries.

An obstacle to the adoption of Meridian in existing services is the need to change the client interface, as clients are often unwilling or unmotivated to update their software. To enable these services to use Meridian, we deployed a DNS to Meridian gateway called ClosestNode.com that allows oblivious clients to perform Meridian lookups by performing DNS resolutions. The gateway also significantly reduces the amount of work required to use and deploy Meridian for both new and existing services.

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

We evaluate Meridian through simulations as well as a deployment on PlanetLab [Bavier et al. 2004]. Our simulations are parameterized by an extensive measurement study, in which we collected node-to-node round-trip latency measurements for 2500 Internet name servers (6.25 million node pairs). We use 500 of these nodes as targets, and the remaining 2000 as overlay nodes in our experiments.

Overall, this paper makes three contributions. First, it outlines a lightweight, scalable, and accurate system for keeping track of location-information for participating nodes. The system is simple, loosely-structured, and entails modest resources for maintenance. The paper shows how Meridian can efficiently find the closest node to a target, the latency minimizing node to a given set of nodes, and the set of nodes that lie in a region defined by latency constraints, which are frequently encountered building block operations in many location-sensitive distributed systems. Although less general than virtual coordinates, we show that Meridian incurs significantly less error. Second, the paper provides a theoretical analysis of our system that shows that Meridian provides robust performance, high scalability and good load balance. This analysis is general and applies to Internet latencies that cannot be accurately modeled with a Euclidean metric. Following a line of previous work on object location (see [Hildrum et al. 2004] for a summary), we give guarantees for the family of growth-constrained metrics. Moreover, we support a much wider family of metrics of low *doubling dimension*. Finally, the paper shows empirical results from both simulations parameterized with measurements from a large-scale network study and a PlanetLab deployment. The results confirm our theoretical analysis that Meridian is accurate, scalable, and load-balancing.

2. FRAMEWORK

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

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

Meridian nodes measure the distance d_j to a peer j, and place that peer in the corresponding ring i such that $r_i < d_j \leq R_i$. This sorting of neighbors into concentric rings is performed independently at each node and requires no fixed landmarks or distributed coordination. Each node keeps track of at most k nodes in each ring and drops peers from overpopulated rings. Consequently, Meridian's space requirement per node is proportional to k. We later show in the analysis (Section 6) that a choice of $k = O(\log N)$ can resolve queries in $O(\log N)$ lookups; in simulations (Section 8), we verify that a small k suffices. We assume that every participating node has a rough estimate of $\log N$.

The ring structure with its exponentially increasing ring radii favors nearby neighbors, enabling each node to retain a relatively large number of pointers to nodes in their immediate vicinity. This allows a node to authoritatively answer geo-

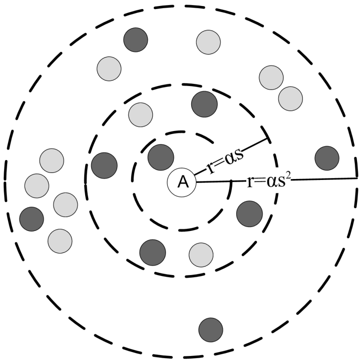

Fig. 1. Each Meridian node keeps track of a fixed number of other nodes and organizes these nodes into concentric, non-overlapping rings of exponentially increasing radii.

graphic queries for its region of the network. At the same time, the ring structure ensures that each node retains a sufficient number of pointers to remote regions, and can therefore dispatch queries towards nodes that specialize in those regions. An exponentially increasing radius also makes the total number of rings per node manageably small and i^* clamps it at a constant.

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

Within a given ring, node choice can have a significant effect on the performance of the system. A set of ring members that are geographically distributed provides much greater utility than a set of ring members that are clustered together, as shown in Figure 1. Intuitively, nodes that are geographically diverse instead of clustered together enable a node to forward a query to a greater region. Consequently, Meridian strives to promote geographic diversity within each ring.

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

We quantify geographic diversity through the hypervolume of the k-polytope formed by the selected nodes. To compute the 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 the same ring, for all $0 \leq i, j \leq k + l$. The coordinates of node i consist of the tuple $\langle d_1^i, d_2^i, ..., d_{k+l}^i \rangle$, where $d_i^i = 0$. This embedding is trivial to construct and does not require a potentially error-introducing mapping from high-dimensional data to a lower number of dimensions.

Having computed the coordinates for all of its members in a ring, Meridian nodes then determine the subset of k nodes that provide the polytope with the largest hypervolume. For small k, it is possible to determine the maximal hypervolume polytope by considering all possible polytopes from the set of $k + l$ nodes. For large $k + l$, evaluating all subsets is infeasible. Instead, Meridian uses a greedy algorithm: A node starts out with the $k+l$ polytope, and iteratively drops the vertex (and corresponding dimension) whose absence leads to the smallest reduction in hypervolume until k vertices remain. The remaining vertices are designated the new primary members for that ring, while the remaining l nodes become secondaries. This computation can be performed in linear time using standard computational geometry tools [Barber et al. 1996]. 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 [Demers et al. 1987] that implements a membership service. The central goal of our gossip protocol is for each node to discover and maintain a small set of pointers to a sufficiently diverse set of nodes in the network. Our gossip protocol works as follows:

(1) Each node A randomly picks a node B from each of its rings and sends a gossip packet to B containing a randomly chosen node from each of its rings.

(2) On receiving the packet, node B determines through direct probes its latency to A and to each of the nodes contained in the gossip packet from A.

(3) After sending a gossip packet to a node in each of its rings, node A waits until the start of its next gossip period and then begins again from step 1.

In step 2, node B sends probes to A and to the nodes in the gossip packet from A regardless of whether B has already discovered these nodes. This re-pinging ensures that stale latency information is updated, as latency between nodes on the Internet can change dynamically. The newly discovered nodes are placed on B's rings as secondary members.

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

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

Maintenance Overhead. The average bandwidth overhead to maintain the multi-resolution rings of a Meridian node is modest. The number of gossip packets a node receives is equal to the number of neighbors ($m \log N$) multiplied by the probability of being chosen as a gossip target by one of the neighbors ($\frac{1}{\log N}$), where m is the number of rings in the ring-set. A node should therefore expect to send and receive m gossip packets and to initiate m^2 probes per gossip period. A node is also the recipient of probes from neighbors of its neighbors. Since it has $m \log N$ neighbors, each of which sends m gossip packets, there are $m^2 \log N$ gossip packets with a $\frac{1}{\log N}$ probability of containing a reference to it. Therefore, a node expects to receive m^2 probes from neighbors of its neighbors. Assuming $m = 9$, a probe packet size of 50 bytes, two packets per probe, and a gossip packet size of 100 bytes, membership dissemination consumes an average of 20.7 KB/period of bandwidth per node. For a gossip period of 60 seconds, the average overhead associated with gossip is 345 B/s, and is independent of system size.

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

3. APPLICATIONS

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

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 [Stoica et al. 2001], Pastry [Rowstron and Druschel 2001] and Tapestry [Zhao et al. 2001], where each hop brings the query exponentially closer to the destination, though in Meridian the routing is performed using physical latencies instead of numerical distances in a virtual identifier space. Another important distinction that Meridian holds over the structured peer-to-peer networks is the target node need not be part of the Meridian overlay. The only requirement is that the latencies between the nodes in the overlay and the target node are measurable. This enables applications such as finding the closest node to a public web server, where the web server is not directly controlled by the distributed application and only responds

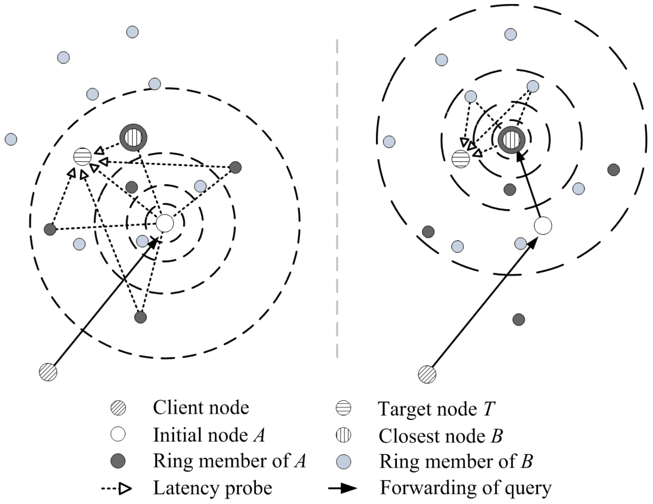

Fig. 2. A client sends a "closest node discovery to target T" request to a Meridian node A, which determines its latency d to T and probes its ring members between $(1-\beta)\cdot d$ and $(1+\beta)\cdot d$ to determine their distances to the target. The request is forwarded to the closest node thus discovered,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

to HTTP queries.

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

Meridian uses an acceptance threshold β, which determines the reduction in distance at each hop. The route acceptance threshold is met if one or more of the queried peers is closer than β times the distance to the target, and the client request is forwarded to the closest node. If no peers meet the acceptance threshold, then routing stops and the closest node currently known is chosen. Figure 2 illustrates the process.

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

Central Leader Election. Another frequently encountered problem in distributed systems is to locate a node that is "centrally situated" with respect to a set of other nodes. Typically, such a node plays a specialized role in the network that requires frequent communication with the other members of the set; selecting a centrally located node minimizes both latency and network load. An example application is

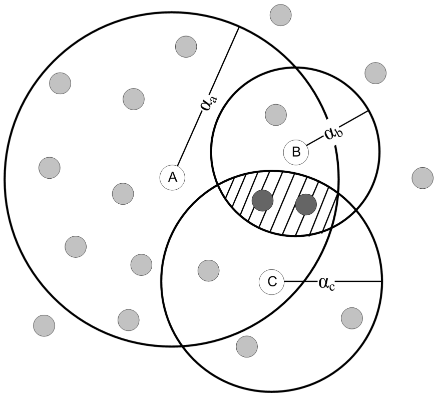

Fig. 3. A multi-constraint query consisting of targets A, B, C with respective latency constraints of $\alpha_a, \alpha_b, \alpha_C$. The shaded area represents the solution space.

leader election, which itself is a building block for higher level applications such as clustering and low latency multicast trees.

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

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

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

Finding a node that satisfies multiple constraints can be viewed as a node selection problem, where the constraints define the boundaries of a region in space (the solution space), as illustrated in Figure 3. A constraint is specified as a target and a latency bound around that target. When a Meridian node receives a multi-constraint query with u constraints specified as $\langle \mathit{target}_i, \mathit{range}_i \rangle$, for all $0 < i \leq u$, it measures its latency d_i to the target nodes and calculates its distance to the solution space as

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

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

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

4. MERIDIAN QUERY LANGUAGE

We described a framework and provided three algorithms for solving three commoningly encountered problems. But there may well be other location-related problems to solve, and other solution strategies that applications may require. To enable such applications that we could not foresee, we added a language for expressing application specific algorithms and a runtime for evaluating such algorithms safely.

The Meridian Query Language (MQL) is a safe, polymorphic, and dynamically typed variant of *C* that provides tight resource and processing constraints on each query. Every Meridian packet carries the full query specified by the user, similar to a capsule in an active network [Tennenhouse and Wetherall 1996], and the query is executed on each Meridian node it is forwarded to, with query forwarding abstracted as simple remote procedure calls.

MQL's grammar and lexical syntax is very similar to *C*. To ensure safety, there are no pointers nor any direct references to memory, and type checking as well as bounds checking is performed at runtime. MQL has the primitives types *int*, *double* and *string*, as well as two primitive structures *Node* and *Measurement*. The two structures are used by many of the library functions that provide access to Meridian operations. The *Node* structure is an abstraction for a Meridian node and contains the address of the node, the Meridian port, and the address and port of an optional rendezvous node used for redirecting traffic to the node if direct connectivity is not available, as is the case for firewalled hosts or hosts behind NATs. The *Measurement* structure is an abstraction for latency information from a source node to a set of targets, and is usually returned as a result of library functions that issue latency probes.

An MQL query is processed in an isolated runtime environment which consists of an interpreter that can multiplex multiple simultaneous queries, and a rich set of native library functions issuing latency probes or accessing the ring structure. The runtime environment enforces local resource constraints, such as the amount of memory or time a query can allocate or execute for per node. Restrictions on per query resources across multiple nodes are also enforced by the runtime environment using auditing information embedded into the query packet header. This allows additional constraints such as the maximum number of hops per query and the query lifetime in the system to be enforced.

4.1 Library Functions

The MQL library interface enables queries to access the underlying Meridian subsystems as well as convenience functions that are commonly used in localization queries. It consists of local functions shown in Figure 4 for accessing local Meridian ring membership information, and performing math and array operations, as well as remote functions shown in Figure 5 for resolving names, issuing probes, and transfering the control flow to another node.

4.2 Closest Node Selection in MQL

MQL enables concise specifications of algorithms for finding desired nodes. Figure 6 illustrates the closest node discovery protocol written in MQL. The MQL version specifies the complete closest node discovery protocol, and is signficantly shorter and easier to understand than our previous hand-crafted C++ version.

5. CLOSESTNODE.COM

There is still substantial effort involved in building and deploying overlays and interacting with the outside world through existing interfaces. To enable new distributed systems to use Meridian easily and to support oblivious clients to perform Meridian lookups, we deployed a DNS to Meridian gateway called ClosestNode.com A registered service of ClosestNode.com, named *dht*, would be given a sub-domain within ClosestNode.com, such as *dht.closestnode.com*. When a client issues a request to resolve *dht.closestnode.com*, our DNS server, which is the authoritative name server for the domain, initiates a closest node discovery on the Meridian overlay specific to the service using the client's DNS server as the target. The IP address of the closest node is then returned to the client.

The clients to the service are oblivious to Meridian as they continue to access it using the DNS interface. The necessary changes to the service itself is minimal. Service providers need to provide the ClosestNode.com DNS server with a small list of nodes that are running their service, and the service needs to either be modified to call a Meridian library function at startup, or start a stand-alone Meridian daemon along-side it. Two PlanetLab services, CobWeb [Song et al. 2005] and OCALA [Joseph et al. 2006], are currently using ClosestNode.com. Both of these

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

Fig. 4. Local system functions for accessing local Meridian ring membership information, and performing math and array operations. The type T in the function definitions is a generic type that can be instantiated as any primitive or abstract data type at runtime.

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

Fig. 5. System functions for issuing remote procedure calls, DNS name resolutions, and latency probes using DNS queries, TCP SYN/ACK packets, ICMP ECHO packets or custom Meridian UDP packets.

```
Measurement closest(double beta, Node target) {
  Node t[] = {target};
  Measurement self = get_distance_tcp(t, -1);
  double self_lat = self.distance[0];
  Node ring_m[] = array_intersect(
    ring_ge((1.0 - beta) * self_lat),
    ring_le((1.0 + beta) * self_lat));
  if (array_size(ring_m) == 0) {
    return self;
  }
  Measurement r_lat[] = get_distance_tcp(ring_m,
    t, ceil((2.0 * beta + 1.0) * self_lat));
  int min_index = -1;
  double min_lat = self_lat;
  for (int i=0; i < array_size(r_lat); i=i+1) {
    double cur_lat = r_lat[i].distance[0];
    if (cur_lat < min_lat) {
      min_index = i;
      min_lat = cur_lat;
    }
  }
  if (min_index == -1) {
    return self;
  }
  Measurement min_n = r_lat[min_index];
  if (min_n.addr != 0
      && min_lat < (self_lat * beta)) {
    Measurement ret_n = rpc(
      ring_m[min_index], closest,
      beta, t);
    if (ret_n.addr != 0) {
      return ret_n;
    }
  }
  return min_n;
}
```

Fig. 6. The closest node discovery protocol in MQL.

services redirect clients to nearby servers, reducing client latency and network load, while improving bandwidth.

6. 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 Meridian 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, our algorithms for nearest neighbor selection and central leader election work well, returning near-exact neighbors and central leaders respectively. We provide further results on *exact* nearest neighbors. Finally, we argue that if the ring sets of different nodes are stochastically independent then the system is load-balanced.

We model the matrix of Internet latencies as a metric, i.e. a symmetric function obeying the triangle inequality. 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 such as assuming 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 [Ng and Zhang 2002; Dabek et al. 2004; Lim et al. 2003; Tang and Crovella 2003; Shavitt and Tankel 2003; Pias et al. 2003; Costa et al. 2004; Ng and Zhang 2004; Lehman and Lerman 2004]. 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 theoretical literature as non-geometric notions of low-dimensionality: *growth-constrained* metrics and *doubling* metrics. Growth-constrained metrics have been considered in the long line of work on DHTs started by Plaxton et al. [Plaxton et al. 1997]. Doubling metrics is a non-trivial generalization of both growth-constrained metrics and low-dimensional Euclidean metrics. In particular, unlike growth-constrained metrics they can 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 do not change with time.

This section is organized as follows. Preliminaries (Section 6.1) are followed by a section on the quality of Meridian rings (Section 6.2). Then we analyze the performance our search algorithms (Section 6.3), with extensions to exact nearest neighbors (Section 6.4) and load-balancing (Section 6.5). We conclude with some directions in which our results can be fine-tuned (Section 6.6). To improve the flow of the paper, some of the more involved proofs are located in the appendices. For completeness, we also include an appendix on tail inequalities (Appendix A).

6.1 Preliminaries

We start with a formal definition of the Meridian system. Let V be the set of all nodes in the system. Nodes running Meridian are called *Meridian nodes.* Let $S_M \subset V$ be the set of Meridian nodes, of size N. Let d be the distance function on V induced by the node-to-node latencies: $d(u, v)$ is the uv-distance, i.e. the latency between nodes u and v. Sometimes, when this is typographically convenient, we may also denote it as 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. Define $B_{ui} = B_u(2^i)$ and $R_{ui} = B_{ui} \setminus B_{(u,i-1)}$. Then R_{ui}'s are disjoint concentric rings around u. Without loss of generality let the smallest distance be 1; denote the maximal distance by Δ.

Throughout this section we will denote the maximal number of nodes in a Meridian ring by k. Formally, for some fixed k every node u maintains $\log(\Delta)$ sets $S_{ui} \subset B_{ui}$, $0 \leq i \leq \lceil \log \Delta \rceil$ of at most k nodes each. These sets are called *m-rings* of u ('m' stands for 'Meridian'), and the nodes in these sets are called *Meridian neighbors* of u. If $|R_{ui}| \geq k$ then the corresponding m-ring S_{ui} consists of exactly k nodes that lie in ring R_{ui}. If $|R_{ui}| < k < |B_{ui}|$ then S_{ui} consists of all nodes in R_{ui}. Finally, if $|B_{ui}| \leq k$ then S_{ui} consists of all nodes in ball B_{ui}.

Let us make some remarks about the above definition. Note that each m-ring S_{ui} contains all Meridian neighbors of u that lie in ring R_{ui}. For a fixed Meridian node u, let i_0 be the largest i such that $B_{ui} \leq k$, and let i_1 be the largest i such that $R_{ui} \leq k$. Then the m-rings S_{ui}, $i \leq i_1$ are fixed by the above definition, whereas the m-rings S_{ui}, $i > i_1$ are not. Also, in the implementation we do not need to maintain m-rings S_{ui}, $i \leq i_0$ explicitely; we define them here for the convenience of the analysis.

Let us formally define the nearest-neighbor search algorithm used in Meridian. Suppose a node u receives a query to a target node t. Then u measures the distance d_{ut} and looks at the three m-rings $S_{(u,i-1)}$, S_{ui} and $S_{(u,i+1)}$, where $i = \lceil \log d_{ut} \rceil$; let S be the union of these rings. All nodes in S measure their distance to t and report their measurements to u. Then u forwards the query to the node $w \in S$ that is closest to the target t subject to the constraint that $d_{ut}/d_{wt} \leq \beta_0$. This constitutes one step of the algorithm.

If such w does not exist, the algorithm chooses the node in $S \cup \{u\}$ that is closest to t, call it w', reports this node to the node that initiated the query, and stops; in this case we say that as a result of the query, our algorithm *finds* w'. Here $\beta_0 > 1$ is a parameter that is the same for all nodes that handle a given query. We denote this algorithm by $\mathcal{A}(\beta_0)$.

For the sake of the analysis we will also consider a version of $\mathcal{A}(\beta_0)$ where instead of looking at three m-rings we look at all m-rings S_{ui}, $i \leq 1 + \lceil \log d_{ut} \rceil$. We denote this version by $\mathcal{A}^*(\beta_0)$.

Let us define the approximation ratio γ for nearest neighbor selection algorithms. Consider a node t and let v be its nearest neighbor. Say node u is a γ-*approximate* nearest neighbor of t if $d_{ut}/d_{vt} \leq \gamma$. An algorithm is γ-*approximate* if for any target it finds a γ-approximate nearest neighbor.

It is straightforward to generalize the algorithms $\mathcal{A}(\cdot)$ and $\mathcal{A}^*(\cdot)$ to the central leader election problem. Namely, given a set T of targets, we simply replace d_{ut}, the distance to from the current node u to target t, by the average distance from u to targets in T. Note that we are back to the nearest neighbor selection problem if $|T| = 1$. The concept of approximation ratio generalizes similarly.

6.2 Quality of the Meridian rings

Intuitively, we want each m-ring S_{ui} to cover the corresponding ring R_{ui} reasonably well: we want each node in R_{ui} to be within a small distance from some node in S_{ui}. For technical reasons in order to cover R_{ui} we might also need some Meridian neighbors from $S_{(u,i-1)}$ or $S_{(u,i+1)}$. We formalize the 'goodness' of m-rings is as follows:

Definition 6.1 *Say the Meridian rings are* ϵ-nice, $\epsilon < 1$, *if for any two Meridian nodes* $u, v \in S_M$ *node* u *has an Meridian neighbor* w *such that* $d(w, v) \leq \epsilon\, d(u, v)$.

In the above definition $v \in R_{ui}$ for $i = \lceil \log d_{uv} \rceil$. Since $2^{i-2} < d_{uw} < 2^{i+1}$, node w is indeed contained in one of the three m-rings $S_{(u,i-1)}$, S_{ui}, $S_{(u,i+1)}$ that are considered by algorithm $\mathcal{A}(\cdot)$.

In Section 6.3 we will how that under Definition 6.1, the Meridian search algorithm achieves good approximation guarantees. Later in this section we show that even for small cardinalities of m-rings it is possible to make them ϵ-nice.

Probabilistic interpretation. To show that the m-rings are indeed ϵ-nice, recall that the m-rings are constructed by an underlying randomized gossiping protocol. For each m-ring S_{ui}, this protocol induces a probability distribution over subsets of S_M, so we can treat S_{ui} as a random variable (whose values are subsets of S_M). In particular, we can talk about the distribution of a given m-ring. A natural and intuitively appealing distribution for an m-ring S_{ui} is that of a random k-node subset of the corresponding ring R_{ui}. Let us formalize this:

Definition 6.2 S_{ui} *is* well-formed *if its distribution is that of a random* k*-node subset of* R_{ui}*, or if* $|R_{ui}| \leq k$.

We proceed to show that if the m-rings are well-formed then even for a small value of k they are ϵ-nice; we model Internet latencies by growth-constrained metrics. Furthermore, we achieve a similar conclusion for a much more general family of doubling metrics.

Growth-constrained metrics. For n-dimensional grid and $\alpha = n + O(1)$, the cardinality of any ball is at most 2^α times smaller than the cardinality of a ball with the same center and twice the radius. This motivates the following definition: the *grid dimension* of a metric is the smallest α such that the above property holds. If the grid dimension is constant, we say that the metric is *growth-constrained*.

Growth-constrained metrics can be seen as generalized grids; they have been used as a reasonable abstraction of Internet latencies in the long line of work on DHTs started by Plaxton et al. [Plaxton et al. 1997] (see the intro of [Hildrum et al. 2004] for a short survey). Growth-constrained metrics have also been considered in the theoretical computer science literature in the context of compact data structures [Karger and Ruhl 2002], routing schemes [Abraham and Malkhi

2005], dimensionality in graphs [Krauthgamer and Lee 2003], and gossiping protocols [Kempe et al. 2001].

We show that even with small ring cardinalities it is possible to make the rings ϵ-nice. We consider 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 a quite unfriendly metric we might be able to choose a relatively well-behaved subset of (Meridian) nodes. We will also assume that the rings are well-formed. Intuitively, this is desirable since in a growth-constrained metric the density is approximately uniform.

Theorem 6.3 *Let the metric on S_M have grid dimension α. Fix $\delta \in (0,1)$ and $\epsilon \leq 1$; let the cardinality of a Meridian ring be $k = O(\frac{1}{\epsilon})^{\alpha} \log(N/\delta)$. Suppose the Meridian rings are created by a random process and are well-formed (but not necessarily independent). Then with probability at least $1-\delta$ they are ϵ-nice.*

Proof. Fix two Meridian nodes u, v. Recall that we are looking for a Meridian neighbor w of node u such that $d_{vw} \leq d_{uv}$. Let $r = \epsilon d_{uv}$ and pick the smallest i such that $d_{uv} + r \leq 2^i$. Then

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

where $\gamma = 4 + 3/\epsilon$. By definition of the grid dimension

$$|B_{ui}| \leq |B_v(\gamma r)| \leq \gamma^{\alpha} |B_v(r)|. \qquad (2)$$

Since $B_u(r)$ lies in $R_{ui} \cup R_{(u,i-1)}$, and the corresponding m-rings S_{ui} and $S_{(u,i-1)}$ are well-formed, at least one node from these two m-rings lands in $B_v(r)$ with some (small) failure probability p. We claim that p is *very* small, namely $p < \delta/N^2$. Indeed, note that p is upper-bounded by the probability of not hitting $B_v(r)$ if we select k nodes uniformly at random from a larger set B_{ui}. By (2) and the Chernoff Bounds[2] the latter probability is at most δ/N^2, claim proved.

Recall that p is a failure probability for a given ordered node pair. By Union Bound, the probability that *any* node pair fails is at most $p \cdot N^2 < \delta$, as required. □

Doubling metrics. Any point set in an n-dimensional Euclidean metric has the following property [Assouad 1983]: for $\alpha = n + O(1)$, every ball of radius r can be covered by 2^{α} balls of radius $r/2$. For an arbitrary metric, we define the *doubling dimension* [Assouad 1983; Gupta et al. 2003] as the smallest α such that the above property holds. If the doubling dimension is constant, we say that the metric is *doubling*.

By definition, doubling metrics generalize low-dimensional Euclidean metrics. This generalization is non-trivial: there exist doubling metrics that are really non-Euclidean. More formally, Gupta et al. [Gupta et al. 2003] proved that there exist doubling metrics on N nodes that need distortion $\Omega(\sqrt{\log N})$ to embed into any Euclidean space.

[2] Chernoff Bounds is a useful fact from Probability: the sum of many bounded independent random variables is close to its expectation with very high probability; see Appendix A for more details.

It is known [Gupta et al. 2003] that the doubling dimension is at most four times the grid dimension, so doubling metrics also subsume growth-constrained metrics. Moreover, there are doubling metrics that are very far from being growth-constrained. One particularly instructive example is the *exponential line*, the subset $\{2^i : \ 0 \leq i \leq N\}$ under the standard metric $d(x, y) = |x - y|$; here the doubling dimension is 1, but the grid dimension is $\log N$. Intuitively, doubling metrics are more powerful than growth-constrained metrics because they can combine very sparse and very dense regions.

Doubling dimension has been introduced in the mathematical literature by Assouad [Assouad 1983] (see [Heinonen 2001] for a thorough mathematical treatment) and has recently become a hot topic in the theoretical computer science community, e.g. [Gupta et al. 2003; Krauthgamer and Lee 2004; Talwar 2004; Kleinberg et al. 2004; Mendel and Har-Peled 2005; Slivkins 2005a]. In particular, it was used to model Internet latencies in the context of distributed algorithms for network embedding and distance estimation [Kleinberg et al. 2004; Slivkins 2005b].

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

Say a measure is *s-doubling* [Heinonen 2001] if for any ball B, the measure of B is at most s times larger than that of a ball with the same center and half the radius. Intuitively, a doubling measure is an assignment of weights to nodes that makes a metric look growth-constrained; for instance, for an N-node exponential line the node with coordinate 2^i will have weight 2^{i-N}. It is known [Heinonen 2001] that for any metric of doubling dimension α there exists a $2^{O(\alpha)}$-doubling measure μ.

With a doubling measure in mind, we extend Definition 6.2 (of well-formed m-rings) as follows:

Definition 6.4 *Consider a measure μ on nodes that assigns a finite non-zero probability to every node. Say than an m-ring S_{ui} is* μ-well-formed *if its distribution is that of a random k-node subset of R_{ui} drawn according to the measure $\mu(\cdot)/\mu(R_{ui})$, or if $|R_{ui}| \leq k$.*

Now we obtain the guarantee in Theorem 6.3 (via a similar proof technique), where instead of well-formed m-rings we use μ-well-formed m-rings, and instead of the grid dimension we plug in a potentially much smaller doubling dimension of S_M.

Theorem 6.5 *Suppose the metric on S_M has doubling dimension α, and let μ be a 2^α-doubling measure on S_M. Fix $\delta \in (0, 1)$ and $\epsilon \leq 1$; let the cardinality of a Meridian ring be $k = (\frac{1}{\epsilon})^{O(\alpha)} \log(N/\delta)$. Suppose the Meridian rings are created by a random process and are μ-well-formed (but not necessarily independent). Then with probability at least $1 - \delta$ they are ϵ-nice.*

Proof. Fix two Meridian nodes uv and let $r = \epsilon d_{uv}$. Pick the smallest i such that $d_{uv} + r \leq 2^i$. By (1), applying the definition of a doubling measure $\log \gamma$ times gives

$$\mu[B_{ui}]/\mu[B_v(r)] \leq 2^{O(\alpha \log \gamma)} = \gamma^{O(\alpha)}. \tag{3}$$

In the proof of Theorem 6.3, we essentially consider the special case when μ is the uniform measure, and use (2) to show that at least one node from S_{ui} or $S_{(u,i-1)}$ lands in $B_v(r)$, with failure probability at most δ/N^2 (and then the theorem follows by the Union Bound). Using (3) instead of (2), this proof trivially generalizes to any μ. □

6.3 Nearest neighbors and central leaders

We prove that the Meridian algorithm for nearest neighbor selection and (more generally) for central leader election achieves good approximation ratios, under the assumption that the Meridian rings are ϵ-nice. Specifically, algorithm $\mathcal{A}(2)$ is 3-approximate, for any $\epsilon \leq \frac{1}{8}$. A better approximation ratio can be proved for algorithm $\mathcal{A}^*(\beta_0)$; the provable accuracy of this algorithm tends to improve as β_0 and ϵ get smaller. The tradeoff between β_0 and the approximation ratio matches our simulation in Section 8 (see Figure 10). We summarize these results as follows:

Theorem 6.6 *Suppose the Meridian rings are ϵ-nice, for some $\epsilon \leq \frac{1}{4}$. Consider Meridian algorithms $\mathcal{A}(2)$ and $\mathcal{A}^*(\cdot)$ for nearest-neighbor search and, more generally, for central leader election. Then:*

(a) algorithm $\mathcal{A}(2)$ is 3-approximate, for any $\epsilon \leq \frac{1}{8}$; completes in $\lceil \log \Delta \rceil$ steps.

(b) algorithm $\mathcal{A}^*(1+\epsilon^2)$ is $(1+3\epsilon)$-approximate; completes in $\lceil \log(\Delta/\epsilon^2) \rceil$ steps.

(c) algorithm $\mathcal{A}^*(1+\gamma)$ is $(1+3\epsilon+\gamma)$-approximate, for any $\gamma \in \left[\epsilon^2; \frac{2}{5}\right]$; completes in $\lceil \log(\Delta/\gamma) \rceil$ steps.

Proof Sketch: Let T be the set of targets, and let $d_T(u)$ be the average distance from node u to the targets in T. Let v^* be the central leader, i.e. the Meridian node that minimizes d_T. For a node u, let $r(u) = d_T(u)/d_T(v^*)$ be the approximation ratio. If the query is forwarded from node u to node v, we say that the *progress* at u is $d_T(u)/d_T(v)$.

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

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

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

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

6.4 Extensions: exact nearest neighbors

We extend our result on growth-constrained metrics (Theorem 6.3 in conjunction with Theorem 6.6) to show that a version of algorithm $\mathcal{A}(2)$ finds *exact* nearest neighbors.

We will use a somewhat more restrictive model: in addition to assuming that the metric on S_M is growth-constrained, we will need a similar assumption about the set $Q \subset V$ of potential targets. Specifically, we consider two settings. In one setting, we assume that the metric on Q is growth-constrained, and that the set S_M of Meridian nodes is chosen uniformly at random from Q. In the other setting we make a more fine-grained assumption: we assume that the metric on $S_M \cup \{q\}$ is growth-constrained, for any target $q \in Q$. Note that here we do not assume that the metric on *all* of Q is growth-constrained; in particular, very dense clusters of potential targets are allowed.

We will show that for any query to a target in Q algorithm $\mathcal{A}(2)$ finds an exact nearest neighbor, and does so in at most $\log(\Delta)$ steps; if this is the case, we say that algorithm $\mathcal{A}(2)$ is Q*-exact.*

Theorem 6.7 *Consider a set $Q \subset V$ of potential targets. Assume either of:*

(a) the metric on Q has grid dimension α, and the set S_M of Meridian nodes is a random N-node subset of Q, or

(b) the metric on $S_M \cup \{q\}$ has grid dimension α, for any node $q \in Q$.

Let $k = 2^{O(\alpha)} \log\left(\frac{1}{\delta} N|Q| \log \Delta\right)$ be the cardinality of each Meridian ring, for a given parameter $\delta > 0$. Suppose the Meridian rings are created by a random process and are well-formed (but not necessarily independent). Then with probability at least $1 - \delta$ the nearest-neighbor selection algorithm $\mathcal{A}(2)$ is Q-exact.

Proof Sketch: Using the technique from Theorem 6.6(a), we prove that the distance to target decreases by a factor of at least 2 on each step except maybe the last one. We have to be careful about this last step, since in general the target is not a Meridian node and therefore not a member of any ring. In particular, this is why bounded grid dimension on just S_M does not suffice.

Part (b) is easier; some extra computation is needed in part (a) due to the fact that there we do not have a good bound on the grid dimension of S_M; instead, we use the assumption that S_M is a random subset of Q. □

6.5 Extensions: load-balancing

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 will use the setting of Theorem 6.7(a), with a further assumption that the m-rings are (stochastically) independent from each other:

Definition 6.8 *Say that the Meridian rings are* independent *if the collection of all m-rings is a collection of independent random variables.*

The above property matches well with our simulation results (see Figure 13).

Theorem 6.9 *Consider a set $Q \subset V$ of nodes and assume that the metric on Q has grid dimension α. Let the set S_M of Meridian nodes be a random N-node subset of Q. For a parameter $\delta > 0$, let the cardinality of a Meridian ring be equal to*

$$k = 2^{O(\alpha)} \log(|Q|/\delta) \log(N) \log(\Delta).$$

Let $\gamma = 2^{O(\alpha)} \log(N\Delta/\delta)$. Suppose the Meridian rings are created by a random process and are well-formed and independent. Then with probability at least $1 - \delta$ the nearest neighbor selection algorithm $\mathcal{A}(2)$ is (γ, Q)-balanced. Recall that it is Q-exact by Theorem 6.7(a).

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

6.6 Fine-tuned versions of the results

Our provable guarantees can be fine-tuned in two directions: to use relaxed versions of the grid dimension, and to rely on average (vs worst-case) guarantees.

First, our results hold under a less restrictive definition of the grid-dimension that only applies to balls that contain sufficiently many nodes: at least $\log(n)$ nodes in Theorem 6.6, and at least $\log(n|Q|)$ nodes in Theorem 6.7 and Theorem 6.9.

Second, the vicinity of a given node u could be significantly more 'well-behaved' than guaranteed by the (global) concept of grid dimension. We can show that in this case some of this node's m-rings can be made smaller. We would like the size of each m-ring S_{ui} to depend only on what happens in the corresponding ball B_{ui}. Specifically, let $r = \epsilon\, 2^{i-3}$ and choose a Meridian node v within distance $2^i - r$ from u such that the ball $B_v(r)$ has the smallest cardinality. Note that $B_v(r) \subset B_{ui}$. Define

$$\sigma_{ui} = |B_{ui}|/|B_v(r)|.$$

Now we can use this ratio, instead of the doubling dimension, to express the 'goodness' of ball B_{ui}. In particular, Theorem 6.3 it suffices to assume that the cardinality of each ring S_{ui} is at least $2.2\, \sigma_{ui} \ln(n^2/\delta)$.

Third, our guarantees are worst-case; *on average* it suffices to query only a fraction of neighbors of a given ring. To take advantage of this observation, we need a minor modification to the search algorithm. 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.

Now in Theorem 6.6(a) (used in conjunction with Theorem 6.3) 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 to find a progress-2 neighbor. Therefore on average on every step (except 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 Theorem 6.6(bc) (used in conjunction with Theorem 6.3), and $k_0 = O(1)^\alpha$ for Theorem 6.7. We obtain similar improvements for Theorem 6.6 used in conjuction with Theorem 6.5 for doubling metrics.

7. EVALUATION

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

Simulation. We performed a large scale measurement study of internode latencies between 2500 nodes to parameterize our simulations. We collected pair-wise round-trip time measurements between 2500 DNS servers at unique IP addresses, spanning 6.25 million node pairs. The study was performed on 10 different PlanetLab nodes, with the median value of the runs taken for the round-trip time of each pair of nodes. Data collection was performed on May 5-13, 2004; query interarrival times were dilated, and the query order randomized, to avoid queuing delays at the DNS servers. The latency measurements between DNS servers on the Internet were performed using the King measurement technique [Gummadi et al. 2002].

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

We compare Meridian to virtual coordinates computed through network embeddings. We computed the coordinates for our 2500 node data set using GNP, Vivaldi and Vivaldi with height [Dabek et al. 2004]. GNP is a global virtual coordinate system based on static landmarks. We configured it for 15 landmarks and 8 dimensions, and used the N-clustered-medians protocol for landmark selection. Vivaldi is a virtual coordinate scheme based on spring simulations and was configured to use 6 dimensions with 32 neighbors. Vivaldi with height is a recent scheme that performs a non-Euclidean embedding which assigns a two dimensional location plus a height value to each node. We randomly select 500 targets from our data set of 2500 nodes.

We first examine the inherent embedding error in absolute coordinate systems

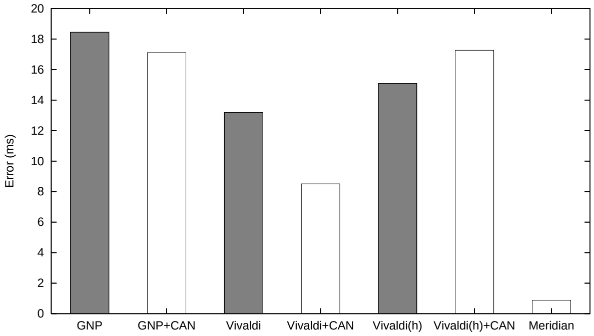

Fig. 7. Light bars show the median error for discovering the closest node. Darker bars show the inherent embedding error with coordinate systems. Meridian's median closest node discovery error is an order of magnitude lower than schemes based on embeddings.

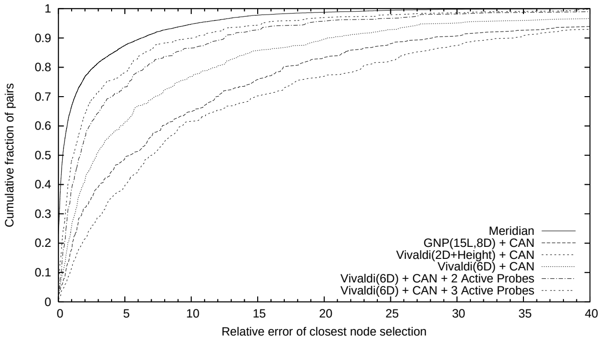

Fig. 8. Meridian's relative error for closest node discovery is significantly better than virtual coordinates.

and determine the error involved in finding the closest neighbor. The dark bars in Figure 7 show the median embedding error of each of the coordinate schemes, where the embedding error is the absolute value of the difference between the measured distance and predicted distance over all node pairs. While these systems incur significant errors during the embedding, they might still pick the correct closest node. To evaluate the error in finding the closest node, we assume the presence of a geographic query routing layer, such as a CAN deployment, with perfect information at each node. This assumption biases the experiment towards virtual coordinate systems and isolates the error inherent in network embeddings. The resulting median errors for all three embedding schemes, as shown by the light bars in Figure 7, are an order of magnitude higher than Meridian. Figure 8 compares the relative error CDFs of different closest node discovery schemes. Meridian has a lower relative error than the embedding schemes by a large margin over the entire distribution.

We also examine the improvement in closest node discovery accuracy using Vi-

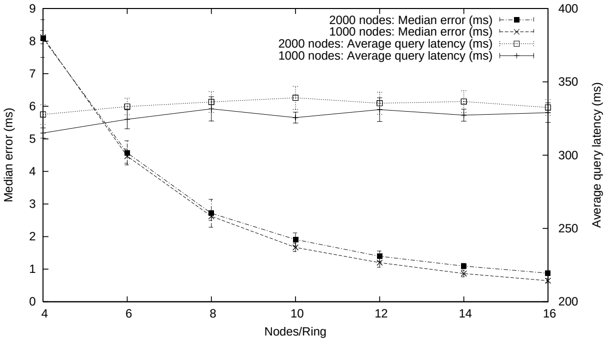

Fig. 9. A modest number of nodes per ring achieves low error. Average latency is determined by the slowest node in each ring and the hop count, and remains constant within measurement error bounds.

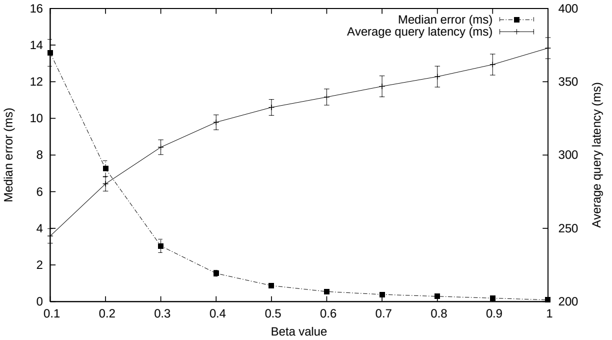

Fig. 10. An increase in β significantly improves accuracy for $\beta \leq 0.5$. The average query latency increases with increasing β, as a bigger β increases the average number of hops taken in a query.

valdi coordinates with the addition of latency data from active probes. We modify Vivaldi+CAN to return the top M candidates based on their coordinates and actively probe the target to determine the closest candidate. Figure 8 shows the results for $M = 2$ and $M = 3$. Active probing greatly improves the accuracy of closest node discovery, but is still significantly less accurate than Meridian. Note that selecting the M closest targets for $M > 1$ in a scalable ($< O(N)$) manner requires additional, complex extensions to CAN that are equivalent to a multi-dimensional expanding ring search.

The accuracy of Meridian's closest node discovery protocol depends on several parameters, such as the number of nodes per ring k, acceptance interval β, the constant α, and the gossip rate. The most critical parameter is the number of nodes per ring k, as it determines the coverage of the search space at each node. Figure 9 shows that median error drops sharply as k increases. Hence, a node only needs to keep track of a small number of other nodes to achieve high accuracy. The

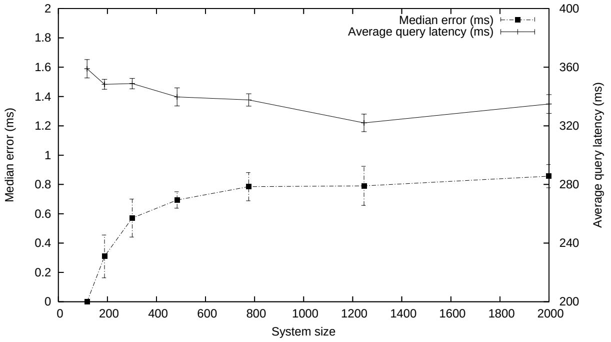

Fig. 11. Median error and average query latency as a function of system size, for $k = \lfloor \log_{1.6} N \rfloor$; both remain constant as the network grows, as predicted by the analytical results.

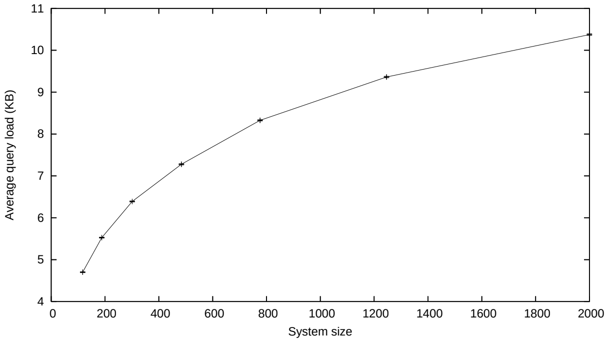

Fig. 12. The average load of a closest node discovery query increases sub-linearly with system size ($k = \lfloor \log_{1.6} N \rfloor$).

results indicate that as few as eight nodes per ring can return very accurate results with a system size of 2000 nodes.

High accuracy must also be coupled with low query latency for interactive applications that have a short lifetime per query and cannot tolerate a long initial setup time. The closest node discovery latency is dominated by the sum of the maximum latency probe at each hop plus the node to node forwarding latency; we ignore processing overheads because they are negligible in comparison. Meridian bounds the maximum latency probe by $2\beta + 1$ times the latency from the current intermediate node to the destination, as any probe that requires more time cannot be a closer node and its result is discarded. The average query latency curve in Figure 9 shows that queries are resolved quickly regardless of k. Average query latency is determined by the hop count and the slowest node in each ring, subject to the maximum latency bound; both increase only marginally as k increases from four to sixteen.

The β parameter captures the tradeoff between query latency and accuracy as

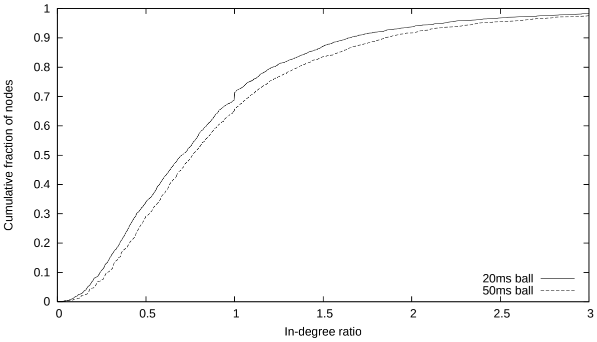

Fig. 13. The in-degree ratio shows the average imbalance in incoming links within spherical regions. More than 90% of regions have a ratio less than 2.

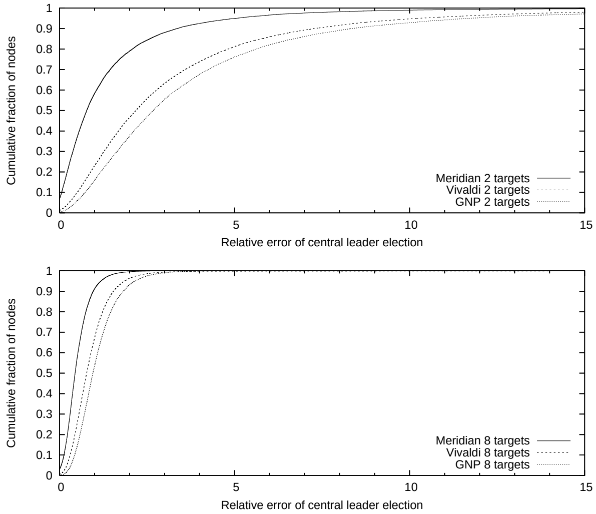

Fig. 14. Central leader election accuracy.

shown in Figure 10. 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 Theorem 6.6). 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 11 plots the median error and average query latency. We set $k = \lfloor \log_{1.6} N \rfloor$ such that the number of nodes per ring varies with the system size; setting k to a constant would favor small system sizes, and this particular log base yields $k = 16$ for 2000 nodes. As predicted by the theoretical analysis in Section 6, the median error remains constant as the network grows, varying only within the error margin. The error

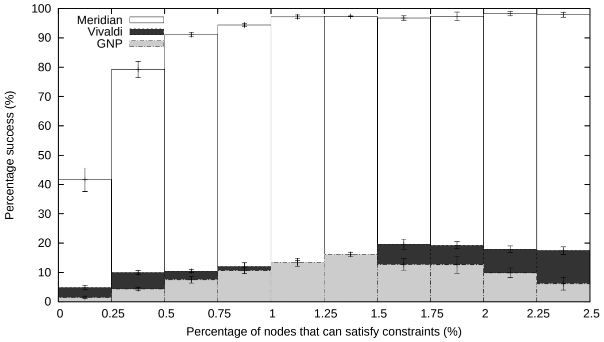

Fig. 15. The percentage of successful multi-constraint queries is above 90% when the number of nodes that can satisfy the constraints is 0.5% or more.

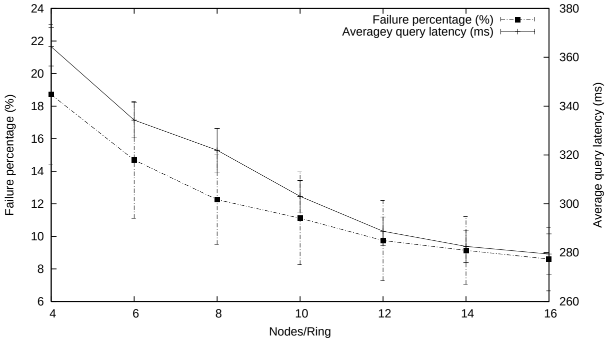

Fig. 16. An increase in the number of nodes per ring k significantly reduces the failure percentage of multi-constraint queries for $k \leq 8$.

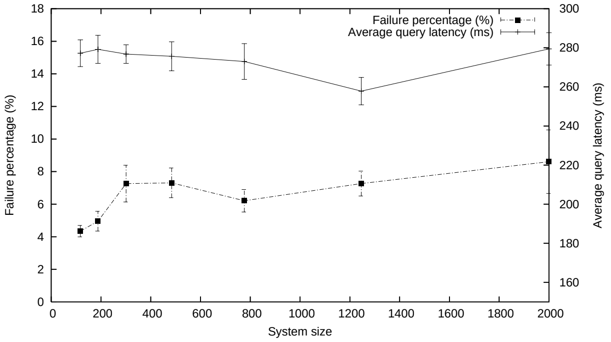

Fig. 17. The percentage of multi-constraint queries that cannot be resolved with Meridian and average query latency. Both are independent of system size.

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 12 plots the total bandwidth required throughout the entire network to resolve a query, that is, the total number of bytes from every packet associated with the query, and shows that it grows sub-linearly with system size, with 2000 nodes requiring a total of 10.4 KB per query.

A desirable property for load-balancing, and one of the assumptions in our theoretical analysis (see Theorem 6.9) is stochastic independence of the ring sets. We verify this property indirectly by measuring the *in-degree ratio* of the nodes in the system. The in-degree ratio is defined as the number of incoming links to a node A over the average number of incoming links to nodes within a ball of radius r around A. If the ring sets are independent, then the in-degree ratio should be close to one; a ratio of one indicates that links to the region bounded by radius r around A are distributed uniformly across the nodes in the area. Figure 13 shows that Meridian distributes load evenly. More than 90% of the balls have an in-degree ratio less than two for balls of radius 20ms and 50ms.

Another useful property, as well as an assumption in our theoretical analysis (see Theorem 6.6), is that ring members are well distributed. To determine the effectiveness of Meridian's ring membership management protocol, we examine the *latency ratio* of the nodes. The latency ratio for a node A and a target node B is defined as the latency of node C to B over the latency of A to B, where C is the neighbor of A that is closest to B. We find that, for $\beta = \frac{1}{2}$, further progress can be made via an extra hop to a closer node more than 80% of the time. For $\beta = 0.9$, an extra hop can be taken over 97% of the time. This indicates that the ring membership management protocol selects a useful and diverse set of ring members. Compared to a random replacement protocol, we find that the standard deviation of relative error is 38ms when using hypervolumes for selection and 151ms when using random replacement; hypervolume-based selection is more consistent and robust.

We evaluate how Meridian performs in central leader election by measuring its relative error as a function of group size. Figure 14 shows that, as group size gets larger, the relative error of the central leader election application drops. Intuitively, this is because the larger group sizes increase the number of nodes eligible to serve as a well-situated leader, and simplify the task of routing the query to a suitable node. Central leader election based on virtual coordinates incurs significantly higher relative error than Meridian for a group size of two. The accuracy gap between coordinate schemes and Meridian closes as the group size increases, as large groups simplify the problem and even random selection becomes competitive with more accurate selection.

We evaluate our multi-constraint protocol by the percentage of queries that it can satisfy, parameterized by the difficulty of the set of constraints. For each multi-constraint query we select four random target nodes and assign a constraint to each target node chosen uniformly at random between 40 and 80 ms. The difficulty of a set of constraints is determined by the number of nodes in the system that can

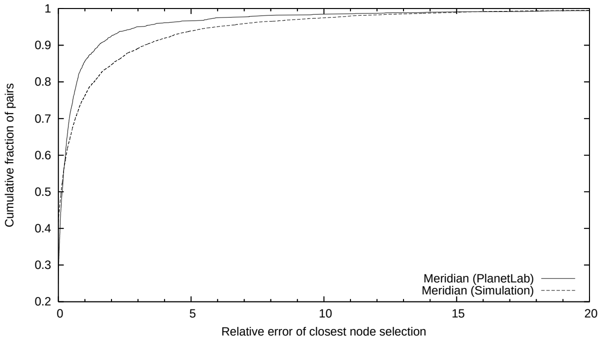

Fig. 18. The relative error of closest node discovery for a Meridian deployment on PlanetLab versus simulation. Meridian achieves results comparable to or better than our simulations in a real-world deployment.

satisfy them. The fewer the nodes that can satisfy the set of constraints, the more difficult is the query.

Figure 15 shows a histogram of the success rate broken down by the percentage of nodes in the system that can satisfy the set of constraints. For queries that can be satisfied by 0.5% of the nodes in the system or more, the success rate is over 90% for Meridian and less than 11% when using coordinate schemes.

As in closest node discovery, k, the number of nodes per ring, has the largest influence on the performance of the multi-constraint protocol. Figure 16 shows that the failure rate decreases as the number of nodes per ring increases. It also shows a decrease in average query latency as the number of nodes per ring increases. An increase in β decreases the failure percentage and increases the average latency of a multi-constraint query, though the performance of the multi-constraint protocol is mostly independent of β.

The scalability properties of the multi-constraint system are very similar to the scalability of closest node discovery. Figure 17 shows that the failure rate and the average query latency are independent of system size. The average load per multi-constraint query (not shown) grows sub-linearly and is approximately four times the average load of closest node discovery query. The non-increasing failure rate and the sub-linear growth of the query load make the multi-constraint protocol highly scalable.

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

Hosts behind firewalls and NATs are very common on the Internet, and a system must support them if it expects large-scale deployment over uncontrolled, heterogeneous hosts. Meridian supports such hosts by pairing each firewalled host with a fully accessible peer, and connecting the pair via a persistent TCP connection. Messages bound for the firewalled host are routed through its fully accessible peer. A ping, which would ordinarily be sent as a direct UDP packet or a TCP connect request, is sent to the proxy node instead, which forwards it to the destination, which then performs the ping to the originating node and reports the result. A node whose proxy fails is considered to have failed, and must join the network from scratch to acquire a new proxy. Since a firewalled host cannot directly or indirectly ping another firewalled host, firewalled hosts are excluded from ring membership on other firewalled hosts, but included on fully-accessible nodes.

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

We deployed the Meridian implementation over 166 PlanetLab nodes. We benchmark the system with 1600 target web servers drawn randomly from the Yahoo web directory, and examine the latency to the target from the node selected by Meridian versus the optimal obtained by querying every node. Meridian was configured with $k = 8$, $s = 2$, $\beta = \frac{1}{2}$, and $\alpha = 1$ms. Overall, median error in Meridian is 1.8ms, and the relative error CDF in Figure 18 shows that it performs better than simulation results from a similarly configured system.

8. RELATED WORK

Meridian is a general network location service that was first introduced in [Wong et al. 2005]. This paper extends the previous work significantly in three directions. First, we describe MQL, an application-specific, safe and expressive query language that provides an extensible way to specify location queries. Second, we describe the integration of our framework with DNS, and describe how client redirection to nearby servers can be done transparently; this is an extension of [Wong and Sirer 2006]. Finally, we show that the system is theoretically sound with a complete theoretical analysis of the system's most common application.

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

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

GNP [Ng and Zhang 2002] determines the coordinates of a node by measuring its latency to a fixed set of landmarks and then solving a multidimensional, iterative, nonlinear minimization problem. ICS [Lim et al. 2003] and Virtual Landmarks [Tang and Crovella 2003] both aim to reduce the computational cost of the GNP embedding algorithm by replacing it with a computationally cheaper, linear approximation based on principal component analysis, though the speedup may incur a loss in accuracy. To avoid the load imbalance and lack of failure resilience created by a set of fixed landmarks, PIC [Costa et al. 2004] and PCoord [Lehman and Lerman 2004] use landmarks only for bootstrapping and calculate their coordinates based on the coordinates of peers. This can lead to compounding of embedding errors over time in a system with churn. NPS [Ng and Zhang 2004] is similar to PIC and PCoord but further imposes a hierarchy on nodes to avoid cyclic dependencies in computing coordinates and to ensure convergence. Lighthouse [Pias et al. 2003] avoids fixed landmarks entirely and uses multiple local coordinate systems that are joined together through a transition matrix to form a global coordinate system.

Simulation-based systems map nodes and latencies into a physical system whose minimum energy state determines the node coordinates. Vivaldi [Dabek et al. 2004] is based on a simulation of springs, and can be augmented with an additional height vector to increase accuracy. Big-Bang Simulation [Shavitt and Tankel 2003] performs a simulation of a particle explosion under a force field to determine node positions.

IDMaps [Francis et al. 2001] is a system that can compute the approximate distance between two IP addresses without direct measurement based on strategically placed tracer nodes. IDMaps incurs inherent errors based on the client's distance to its closest tracer server and requires deploying system wide infrastructure. Other work [Fei et al. 1998] has also examined how to delegate probing to specialized nodes in the network.

Recent theoretical work [Kleinberg et al. 2004; Slivkins 2005b] has sought to explain the empirical success of network embeddings and IDMaps-style approaches.

Server Selection: Our closest node discovery protocol draws its inspiration from the Chord DHT [Stoica et al. 2001], which performs routing in a virtual identifier space by halving the virtual distance to the target at each step. Proximity based neighbor selection [Castro et al. 2002; 2003] populates DHT routing tables with nearby nodes, which decreases lookup latency, but does not directly address location-related queries. The time and space complexity of two techniques are discussed in [Hildrum et al. 2002] and [Karger and Ruhl 2002], but these techniques focus exclusively on finding the nearest neighbor, apply only to Internet latencies modeled by growth-constrained metrics, and have not been evaluated with a large scale Internet data.

In beaconing [Kommareddy et al. 2001], landmark nodes keep track of their latency to all other nodes in the system. A node finds the closest node by querying all landmarks for nodes that are roughly the same distance away from the landmarks. This approach requires each landmark to retain $O(N)$ state, and can only resolve nearest neighbor queries. Binning [Ratnasamy et al. 2002] operates similarly, using approximate bin numbers instead of direct latency measurements. Mithos [Waldvogel and Rinaldi 2002] provides a gradient descent based search protocol to find proximate neighbors in its overlay construction. It is similar to Meridian as it is iterative and performs active probing but it requires $O(N)$ hops to terminate. It is also more prone to terminate prematurely at a local minimum than Meridian as it does not promote diversity in its neighbor set. Various active-probing based nearest neighbor selection schemes are proposed in [Shanahan and Freedman 2005]. These schemes require $O(N)$ state per node, which limits their scalability, and are non-trivial to adapt to other positioning problems. Tiers [Banerjee et al. 2002] reduces the state requirement by forming a proximity-aware tree and performing a top-down search to discover the closest node. Hierarchical systems suffer inherently from load imbalance as nodes close to the root of the hierarchy service more queries, which limits scalability when the workload increases with system size.

Early work on locating nearby copies of replicated services [Guyton and Schwartz 1995] examined combining traceroutes and hop counts to perform a rough triangulation, and to determine the closest replica at a centralized $O(N)$ server using Hotz's distance metric [Hotz 1994]. Dynamic server selection was found in [Carter and Crovella 1997] 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 [Carter and Crovella 1999] 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.

9. CONCLUSIONS

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

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

We have argued analytically that Meridian provides robust performance, delivers high scalability, and balances load evenly across nodes. We have evaluated our system through a PlanetLab deployment as well as extensive simulations, parameterized by data from measurements of 2500 nodes and 6.25 million node pairs. The evaluation indicates that Meridian is effective; it incurs less error than systems based on an absolute embedding, is decentralized, requires relatively modest state and processing, and locates nodes quickly. We have deployed a DNS to Meridian gateway that enables oblivious clients to issue Meridian lookups, reducing the amount of work required to use Meridian. We have shown how the framework can be used to solve three network positioning problems frequently-encountered in distributed systems, and described a domain specific language that can be used to express other application-specific algorithms. It remains to be seen whether the lightweight approach advocated in this paper can be applied to other significant problems.

Acknowledgments

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

us the Vivaldi and GNP software and data.

REFERENCES

Abraham, I. and Malkhi, D. 2005. Name independent routing for growth bounded networks. In *17th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*. 49–55.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Heinonen, J. 2001. *Lectures on analysis on metric spaces.* Universitext. Springer-Verlag.

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

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

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

Hotz, S. 1994. Routing Information Organization to Support Scalable Interdomain Routing with Heterogeneous Path Requirements. Ph.D. thesis, Univ. of Southern California.

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

Joseph, D., Kannan, J., Kubota, A., Lakshminarayanan, K., Stoica, I., and Wehrle, K. 2006. OCALA: An Architecture for Supporting Legacy Applications over Overlays. In *NSDI.* San Jose, CA.

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

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

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

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

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

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

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

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

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

Mendel, M. and Har-Peled, S. 2005. Fast construction of nets in low dimensional metrics, and their applications. In *21st ACM Symposium on Computational Geometry (SoCG)*. 150–158.

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

NG, T. AND ZHANG, H. 2002. Predicting Internet Network Distance with Coordinates-Based Approaches. In *INFOCOM*. New York, NY.

NG, T. AND ZHANG, H. 2004. A Network Positioning System for the Internet. In *USENIX*. Boston, MA.

PIAS, M., CROWCROFT, J., WILBUR, S., HARRIS, T., AND BHATTI, S. 2003. Lighthouses for Scalable Distributed Location. In *Intl. Workshop on Peer-To-Peer Systems*. Berkeley, CA.

PLAXTON, C., RAJARAMAN, R., AND RICHA, A. 1997. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In *Symposium on Parallel Algorithms and Architectures*. Newport, RI.

RATNASAMY, S., FRANCIS, P., HADLEY, M., KARP, R., AND SHENKER, S. 2001. A Scalable Content-Addressable Network. In *SIGCOMM*. San Diego, CA.

RATNASAMY, S., HANDLEY, M., KARP, R., AND SHENKER, S. 2002. Topologically-Aware Overlay Construction and Server Selection. In *INFOCOM*. New York, NY.

ROWSTRON, A. AND DRUSCHEL, P. 2001. Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *Middleware*. Heidelberg, Germany.

SHANAHAN, K. AND FREEDMAN, M. 2005. Locality Prediction for Oblivious Clients. In *Intl. Workshop on Peer-To-Peer Systems*. Ithaca, NY.

SHAVITT, Y. AND TANKEL, T. 2003. Big-Bang Simulation for Embedding Network Distances in Euclidean Space. In *INFOCOM*. San Francisco, CA.

SLIVKINS, A. 2005a. Distance Estimation and Object Location via Rings of Neighbors. In *Symposium on Principles of Distributed Computing*. Las Vegas, NV.

SLIVKINS, A. 2005b. Distributed Approaches to Triangulation and Embedding. In *the Symposium on Discrete Algorithms*. Vancouver, BC, Canada.

SONG, Y., RAMASUBRAMANIAN, V., AND SIRER, E. 2005. Optimal Resource Utilization in Content Distribution Networks. In *Computing and Information Science Technical Report TR2005-2004*. Cornell University.

STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, F., AND BALAKRISHNAN, H. 2001. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *SIGCOMM*. San Diego, CA.

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

TANG, L. AND CROVELLA, M. 2003. Virtual Landmarks for the Internet. In *Internet Measurement Conference*. Miami, Florida.

TENNENHOUSE, D. AND WETHERALL, D. 1996. Towards an Active Network Architecture. *Computer Communication Review 26,* 2.

WALDVOGEL, M. AND RINALDI, R. 2002. Efficient Topology-Aware Overlay Network. In *Hot Topics in Networks*. Princeton, NJ.

WONG, B. AND SIRER, E. 2006. ClosestNode.com: An Open-Access, Scalable, Shared Geocast Service for Distributed Systems. *SIGOPS Operating Systems Review 40,* 1.

WONG, B., SLIVKINS, A., AND SIRER, E. 2005. Meridian: A Lightweight Network Location Service without Virtual Coordinates. In *SIGCOMM*. Philadelphia, PA.

ZHAO, B., KUBIATOWICZ, J., AND JOSEPH, A. 2001. Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing. In *Technical Report UCB/CSD-01-1141*. UC Berkeley.

A. TAIL INEQUALITIES

Here we describe the tail inequalities used in the proofs in Section 6. Essentially, a *tail inequality* is a statement that the sum of random variables is very close to its expectation with very low failure probability. We start with the standard *Chernoff Bounds*:

Lemma A.1 (Chernoff Bounds [Motwani and Raghavan 1995]) *Consider the sum X of n independent random variables $X_i \in [0, y]$.*

(a) for any $\mu \leq E(X)$ and any $\epsilon \in (0,1)$ we have $\Pr[X < (1-\epsilon)\mu] \leq exp(-\epsilon^2\mu/2y)$.

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

We also use two easy applications of Chernoff Bounds. Their proofs are fairly standard; we include them here for the sake of completeness.

Claim A.2 *Consider sets $T \subset V$. Suppose we choose a k-node subset $S \subset V$ uniformly at random from V. Then with failure probability at most $e^{-(1-1/\mu)^2\mu/2}$ some node from S lands in T, where $\mu = k|T|/|V|$.*

Proof. Denote the desired event by A. The distribution of S_{ui} is that of the following process P: pick nodes from V independently and uniformly at random, until we gather k distinct nodes. For simplicity consider a slightly modified process P': pick k 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 $\Pr[X_j = 1] = |T|/|V|$, so $\mu = E(\sum X_j)$. The claim follows from Lemma A.1(a) with $y = 1$ and $1 - \epsilon = 1/\mu$. □

Claim A.3 *Consider two sets $S' \subset S$ and suppose n nodes are chosen independently and uniformly at random from S; say X of them 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 equals 1 if and only if the j-th chosen node lands in S'. Then $X = \sum_{j=1}^n X_j$ is a sum of independent random variables, so

$$E(X) = n \cdot \Pr[X_j = 1] = n \cdot |S'|/|S| = \lambda.$$

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

B. FULL PROOF OF THEOREM 6.6 ON CENTRAL LEADER ELECTION

Let us recap the definitions from the proof sketch. Let use fix the set of targets T, and let $d_T(u)$ be the average distance from node u to the targets in T. Let v^* be the central leader, i.e. the Meridian node that minimizes d_T. For a node u, let $r(u) = d_T(u)/d_T(v^*)$ be the approximation ratio.

Recall that if the query is forwarded from node u to node v, we say that the *progress* at u is $d_T(u)/d_T(v)$. More generally, if node v is a Meridian neighbor of node u, say that v is a *progress-β* neighbor, for $\beta = d_T(u)/d_T(v)$. We will use a function

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

Note that for $\beta \in (1,\, 1/\epsilon)$ this function is continuously increasing to infinity.

The following claim captures the performance of a single step of the central leader election algorithm.

Claim B.1 *Assume the rings are ϵ-nice, $\epsilon \le 1/3$. Let u be any Meridian node, and suppose $r_T(u) = f_\epsilon(\beta)$ for some $\beta \in (1, \frac{1}{\epsilon})$. Then a progress-$\beta$ neighbor of u exists and is found by the algorithm $\mathcal{A}^*(\beta)$. Moreover, if $\beta = 2$ then such neighbor is found by algorithm $\mathcal{A}(2)$ as well.*

Proof. First we claim that such neighbor exists. Indeed, pick the smallest i such that $d(u,v^*)(1+\epsilon) \le 2^i$. Since the rings are ϵ-nice, node u has a Meridian neighbor w within distance $\epsilon\, d(u,v^*)$ from node v^*. Then

$$\begin{aligned} d_T(w) &\le d_T(v^*) + d(w,v^*) \le d_T(v^*) + \epsilon\, d(u,v^*) \\ &\le d_T(v^*) + \epsilon\ (d_T(u) + d_T(v^*)) \\ &\le \epsilon\, d_T(u) + (1+\epsilon)\, d_T(u)/f_\epsilon(\beta) \\ &= d_T(u)/\beta, \end{aligned}$$

claim proved.

It is easy to see that w lies in $S_{ui} \cup S_{(u,i-1)}$. To prove that node w is found by $\mathcal{A}^*(\beta)$ it suffices to show that both m-rings are considered by this algorithm, i.e. that $i \le 1 + \lceil \log d_T(u) \rceil$. Indeed,

$$\begin{aligned} d(u,v^*) &\le d_T(u) + d_T(v^*) \le d_T(u)\ \left(1 + f_\epsilon(\beta)^{-1}\right) \\ &\le 2\, d_T(u)/(1+\epsilon), \\ 2^i &< 2\, d(u,v^*)\,(1+\epsilon) \le 4\, d_T(u). \end{aligned}$$

Finally, for the case $\beta = 2$ we need to show that node w is found by algorithm $\mathcal{A}(2)$ as well. Specifically, we need to prove two things:

(i) if $w \in S_{(u,i-1)}$ then $i - 1 \ge j - 1$.
(ii) if $w \in S_{ui}$ then $i \ge j - 1$.

First let us note that by the triangle inequality we have

$$d(u,w) \ge d_T(u) - d_T(w) \ge d_T(u)/2. \tag{4}$$

Now if $w \in S_{(u,i-1)}$ then $d_T(u) \leq 2\,d(u,w) \leq 2^i$ by (4); it follows that $i \geq j$, proving (i). For (ii) recall that

$$d(u,w) \leq d(u,v) + d(v,w) \leq (1+\epsilon)\,d(u,v) \leq 2^i. \tag{5}$$

By (4), (5) and the definition of j it follows that

$$2^j \leq 2\,d_T(u) \leq 4\,d(u,w) \leq 2^{i+2},$$

so $j \leq i+1$, proving (ii). □

Let us state some properties of the function $f_\epsilon(\beta)$ that will be used in the forthcoming proof of Theorem 6.6. Out of these five properties, most crucial is property (c): in conjunction with Claim B.1 it shows that in one step our search algorithm passes from an $f_\epsilon(1+\gamma)$-approximate neighbor to an $f_\epsilon(1+\gamma/2)$-approximate neighbor.

Claim B.2 *Some useful properties of the function $f_\epsilon(\beta)$:*

(a) function $f_\epsilon(2)$ is at most 8 whenever $\epsilon \leq \frac{1}{3}$, and at most 3 whenever $\epsilon \leq \frac{1}{8}$.
(b) $f_\epsilon(1+\gamma)/(1+\gamma) \leq f_\epsilon(1+\gamma/2)$, for any $\epsilon \leq \frac{1}{3}$ and any $\gamma \in (0,1)$.
(c) $f_\epsilon(1+\epsilon^2/2) \leq 1+3\epsilon$ for any $\epsilon \leq \frac{1}{4}$.
(d) $f_\epsilon(1+\gamma/2) \leq 1+3\epsilon+\gamma$, for any $\epsilon \leq \frac{1}{4}$ and any $\gamma \in (0,\frac{2}{5})$.

Proof Sketch: Part (a) are trivial: just plug in the definition of $f_\epsilon(2)$. For parts (bcd), we plug in the definition of $f_\epsilon(\cdot)$ and carefully solve the resulting inequality for ϵ.

In part (b) the inequality reduces to $\epsilon \leq 1/(2+\gamma)$, which holds for any $\epsilon \leq \frac{1}{3}$.

In part (c) we get $g(\epsilon) := \epsilon((1+3\epsilon)^2+20) \leq 6$, which is true for any $\epsilon \leq \frac{1}{4}$ since the function $g(\epsilon)$ is increasing in ϵ and $g(\frac{1}{4}) < 6$.

Finally, in part (d) the inequality reduces to

$$g(\epsilon) := \epsilon^2(3\gamma+6) + \epsilon(\gamma^2+4\gamma-2) - \gamma \leq 0.$$

Since $g(0) = -\gamma < 0$ and the polynomial $g(\epsilon)$ is a quadratic in ϵ, it has two roots, call them ϵ_1 and ϵ_2, and it is negative for any $\epsilon \in (\epsilon_1;\epsilon_2)$. Therefore it suffices to show that $g(\frac{1}{4}) < 0$. Indeed, solving the latter inequality for γ we get $\gamma < (\sqrt{41}-3)/8$, which is more than $\frac{2}{5}$. □

Now we are ready to prove Theorem 6.6.

Proof of Theorem 6.6(a): We need to prove that algorithm $\mathcal{A}(2)$ finds a 3-approximate neighbor of q. By Claim B.1 while the query visits nodes u such that $r_T(u) \geq f_\epsilon(2)$, the algorithm finds a progress-2 neighbor of u and forwards the query to it. The distance $d_T(u)$ 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 $r(v)$ is less than $f_\epsilon(2)$, which is at most 3 by Claim B.2(a). □

Proof of Theorem 6.6(b): We will show that $\mathcal{A}^*(\beta)$ finds a $(1+3\epsilon)$-approximate neighbor of q. The query proceeds in two stages. In the first stage, while the query visits nodes u such that $r_T(u) \geq f_\epsilon(2)$, by Claim B.1 the distance $d_T(u)$ 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 $r(v)$ is less than $f_\epsilon(2)$, which is at most 8 by Claim B.2(a).

In the second phase the progress at each step is smaller than 2. Specifically, by Claim B.1 and Claim B.2b our search algorithm passes from an $f_\epsilon(1+\gamma)$-approximate central leader to an $f_\epsilon(1+\gamma/2)$-approximate central leader, for any $\gamma \in (0,1)$. By induction on i we show that after i more steps the query will arrive at node w such that $r(w) < f_\epsilon(1+2^{-i})$. So $i = \lceil \log(2/\epsilon^2) \rceil$ steps suffices by Claim B.2c. □

Proof of Theorem 6.6(c): The proof is similar to that for part (b); in the second stage $i = \lceil \log 2/\gamma \rceil$ steps suffices by Claim B.2d. □

C. FULL PROOF OF THEOREM 6.7 ON EXACT NEAREST NEIGHBORS

We start with part (b) since it is simpler.

Proof of Theorem 6.7(b): Let the size of a Meridian ring be $k = 2.2{\cdot}10^\alpha \ln(1/p)$, where $p = \delta/N|Q|\log(\Delta)$. 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 the smallest i such that $1.5d \leq 2^i$.

We claim that either $v \in S_{ui}$, or with failure probability at most p node u has a Meridian neighbor $w \in B_q(d/2)$. Indeed,

$$\begin{aligned} B_{ui} &\subset B_q(2^i + d) \subset B_q(4d) \\ |B_{ui}| &\subset |B_q(4d)| \leq 8^\alpha\, |B_q(d/2)|, \end{aligned}$$

so if $|B_{ui}| \geq k$ then the claim follows from Claim A.2; 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.

Recall that, letting $j = \lceil \log d \rceil$, algorithm $\mathcal{A}(2)$ at node u considers the m-rings $S_{(u,j-1)}$, S_{uj} and $S_{(u,j+1)}$. Since by the triangle inequality $d/2 \leq d_{uw} \leq 3d/2$, node w lies in one of these three m-rings, and therefore is found by $\mathcal{A}(2)$. So the progress is at least 2 at every step except maybe the last one, with failure probability at most p. Therefore the algorithm makes at most $\log\Delta$ steps before completion.

Finally, for a single (u,q) pair the failure probability for a single step is at most p. Taking the Union Bound over all $N|Q|$ possible (u,q) pairs and all $\lceil \log\Delta \rceil$ possible steps, it follows that the total probability is at most δ. □

Theorem 6.7(a) is proved using the same idea, except we need to address the fact that Meridian nodes themselves are chosen at random from Q.

Proof of Theorem 6.7(a): 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. Denote $Q_{ui} = Q_u(2^i)$ and let the cardinality of a Meridian ring be

$$k = 8 \cdot 8^\alpha \ln\left(\tfrac{2}{\delta}\, N|Q| \log\Delta\right). \tag{6}$$

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 $B = B_q(d/2)$; choose the smallest i such that $1.5\, d \leq 2^i$.

Note that without loss of generality we can view the process of selecting S_M from Q as follows: choose the cardinality x for B_{ui} from the appropriate distribution, then choose, independently and uniformly at random, x nodes from Q_{ui}, and $n-x$ nodes from $Q \setminus Q_{ui}$.

We claim that with failure probability at most $\delta' = \delta / N|Q| \log(\Delta)$ either $v \in S_{ui}$, or node u has a Meridian neighbor $w \in B$. 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 by the triangle inequality $Q_{ui} \subset Q_q(2^i + d) \subset Q_q(4d)$, it follows that

$$\frac{x}{E(|B|)} = \frac{|Q_{ui}|}{|Q_u(d/2)|} \leq \frac{|Q_u(4d)|}{|Q_u(d/2)|} \leq 8^{\alpha},$$

where the last inequality holds by definition of the grid dimension. Therefore by Claim A.3(a) with failure probability at most $\delta'/2$ the cardinality of B is at least half the expectation. If it is indeed the case that, then by Claim A.2 with failure probability at most $\delta'/2$ some node in ring S_{ui} lands in B. So the total failure probability is at most δ', claim proved.

As in the proof of part (b), we show that node w is found by algorithm $\mathcal{A}(2)$. Therefore the progress is at least 2 at every step except maybe the last one, with failure probability at most δ'. Finally, we take the Union Bound over all $N|Q|$ possible (u, q) pairs and all $\log \Delta$ possible steps to show that the probability that any such pair fails on any step is at most δ. □

D. FULL PROOF OF THEOREM 6.9 ON LOAD-BALANCING

In this section we will prove Theorem 6.9 on load-balancing. A large part of the proof is the setup (Sections D.1 and D.2): 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 technical reasons we introduce some minor changes in the definition of the m-rings and in the search algorithm; these changes do not (really) affect the practical implementation of Meridian. Proving our result for the exact version of Meridian that is implemented leads to mathematical difficulties that are far beyond the scope of this paper.

Recall that for the present theorem we use the setting of Theorem 6.7(a). Compared to the latter, we increase the ring cardinalities by a factor of $O(\log N)(\log \Delta)$. This is essentially because we cannot use Chernoff Bounds on collections of random variables that are *almost* independent – we need exact independence, which is hard to come by. We conjecture that this blow-up can be avoided by a more careful analysis of almost-independent random variables. However, such analysis is again beyond the scope of this paper.

D.1 Setup: Meridian rings and the search algorithm

For convenience, for any $x > 0$ let us define a set of integers $[x] = \{0, 1 \ldots \lceil x \rceil\}$.

Recall that each m-ring S_{ui} was defined as a subset of the corresponding ring R_{ui}, as long as $|B_{ui}| \geq k$. Here to simplify the proofs let us allow each S_{ui} to be an arbitrary subset of B_{ui}:

Definition D.1 *The distribution of each Meridian ring S_{ui} is the distribution of* *a set of k nodes that are drawn independently and uniformly at random from the corresponding ball B_{ui}, possibly with repetitions.*

Note that all previous results for growth-constrained metrics work under this definition as well.

Recall that on every step in algorithm $\mathcal{A}(\cdot)$ 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. We will relax this as follows: 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 $\mathcal{A}(\cdot)$ carry over to this modification.

We will now proceed to define a specific version of $\mathcal{A}(2)$ which can be seen as a rule to select between different progress-2 nodes; we denote it $\mathcal{A}$.

Recall that each ring S_{ui} consists of k nodes from B_{ui}. More formally, let us say that S_{ui} consists of k *slots*, each of which is a node id selected independently and uniformly at random from B_{ui}. Let us partition these slots into $L\log(\Delta)$ collections of size k' each, where

$$\begin{aligned} L &= 6\ln\left(\tfrac{1}{\delta}N\log\Delta\right), \\ k' &= 8\cdot 10^{\alpha}\ln(2K/\delta), \\ K &= N|Q|L\log(\Delta). \end{aligned}$$

We will denote these collections by $\mathcal{C}_{ui}(j,l)$, where $j \in [\log\Delta]$ and $l \in [L]$. Each collection will just consist of k' consecutively numbered slots, starting from the slot number $(jL+l)k'$. Let $S_{ui}(j,l)$ be the set of nodes whose ids are stored in the slots in collection $\mathcal{C}_{ui}(j,l)$. Obviously, $S_{ui}(j,l) \subset B_{ui}$, and 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 (essentially) lets algorithm $\mathcal{A}(2)$ handle this query using only the neighbors in $S_{ui}(j,l)$, for the corresponding i. Specifically, node u sets $i = 1+\lfloor\log d_{uq}\rfloor$ and asks every node in $S_{ui}(j,l)$ to measure the distance to q. Out of these nodes, let w be one that is closest to q. If w is a progress-2 node, then the query is forwarded to w; else, the search stops, and node w is reported to the node that originated the query.

Using the argument from part (a) we can show that for a given tuple (u,q,j,l) either the corresponding set $S_{ui}(j,l)$ 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 at most δ.

Note that algorithm $\mathcal{A}$ can be seen as $\mathcal{A}(2)$ with a rule to select between different progress-2 nodes if such nodes exist: namely, choose a progress-2 node from the corresponding $S_{ui}(j,l)$.

D.2 Setup: randomization and random variables

Recall that each $S_{ui}(j,l)$ is a set of k' nodes drawn from B_{ui} independently and uniformly at random, possibly with repetitions. Moreover, once the set S_M of all Meridian nodes is fixed then (since the m-rings are independent), the collection of all sets

$$\{S_{ui}(j,l):\; u \in S_M,\, i,j \in [\log\Delta], l \in [L]\}$$

is a collection of independent random variables.

We consider the probability distribution induced by several independent random choices, namely:

- a random N-node subset S_M of Q,
- random subsets $S_{ui}(j,l) \subset B_{ui}$, independently 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 the following order of events. First, S_M and t_u's are chosen. After that the time proceeds in $\log(\Delta)$ epochs. In a given epoch j, all subsets $S_{ui}(j,\cdot)$ are chosen, and then all queries are advanced for one step.

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; here j ranges between 0 and $\log\Delta$:

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

Note that $X_{uv}(j,l) \leq X_u^j/L$ and $Y_{uv}(j,l) \leq X_u^{j-1}/L$.

D.3 The actual proof

First let us 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 $\psi = N/|Q|$. By Claim A.3 $|B_u(r)|$ and $|T_u(r)|$ are close to its expectation:

Claim D.2 *With failure probability at most δ, for any $u \in S_M \cup t(T)$ and radius r the following holds:*

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

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 (γ, Q)-balanced if and only if

$$\textstyle\sum_j (k' X_u^j + Y_u^j) \leq 2\gamma k' \log(\Delta) \tag{7}$$

for every node u. Recall that γ is a parameter in the theorem statement.

Definition D.3 *Property* $\mathbf{P(j)}$ *holds if and only if for each node* v *it is the case that* $X_v^j \leq \gamma$ *and* $Y_v^j / k' \leq \gamma$.

By (7) it suffices to prove that with high probability $\mathbf{P(j)}$ holds for all j; recall that j ranges between 0 and $\log \Delta$. It suffices to prove the following inductive claim:

Claim D.4 *If property* $\mathbf{P(j-1)}$ *holds, then with failure probability at most* $\delta / \log(\Delta)$ *property* $\mathbf{P(j)}$ *holds, too.*

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

Let's prove Claim D.4. 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 node v the random variables

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

are independent, and so are the random variables

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

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

Claim D.5 *For every Meridian node* v *and every step* j *we have* **(a)** $E(X_v^j) \leq \gamma/2$ *and* **(b)** $E(Y_v^j / k') \leq \gamma/2$.

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 $y = \gamma/L$. Applying Claim A.1(b) with $\mu = \gamma/2$, we see that

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

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.1(b) we can upper-bound $\Pr[Y_v^j / k' > \gamma]$. By the Union Bound property $P(j)$ holds with the total failure probability at most δ. This completes the proof of Claim D.4.

It remains to prove Claim D.5. Let S_0 be the set of queries $q \in T$ such that v is a nearest neighbor of the target $t(q)$.

Claim D.6 $|S_0| \leq O(2^\alpha)\log(N/\delta)$.

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 D.2 $|Q_t(d/\tau)| \leq O(\log(n/\delta))$. 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 D.2. 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 D.7 *For any query $q \in T \setminus (S_0 \cup R_0)$, letting $t = t(q)$, 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 D.2 we have

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

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 $r = r_0\, 2^i$

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

This completes the proof of Claim D.5(a). For Claim D.5(b), 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 D.2. The rest of the proof of Claim D.5(b) is similar to that of Claim D.5(a). This completes the proof of Claim D.5 and Theorem 6.9.