MCR: Structure-Aware Overlay-Based Latency-Optimal Greedy Relay Search

Yongquan Fu and Ernst Biersack

***Abstract*— Geo-distributed network applications typically use relays to process and forward timely messages among clients. The state-of-the-art approaches greedily locate a relay that is closer to clients based on an overlay that favors neighbors in the immediate vicinity of the current node. Unfortunately, as clients are unknown *a priori*, the optimal relay is generally outside of the immediate vicinity of the current node. Consequently, the search process often terminates at a poor local minimum. In this paper, we address these challenges by designing and implementing a distributed relay-search system called MCR. In order to accurately locate a relay closer to clients, by observing that the latency space exhibits a proximity clustering phenomenon where nodes in the same cluster are typically within close proximity, we propose an overlay called MCRing that is aware of global proximity clusters. In order to scale well under dynamic relays, we maintain the proximity clusters via a gossiping-based clustering process. Furthermore, we propose a series of algorithms to accurately locate a relay that is closer to clients and satisfies the load constraints. We prove that the relay-search process achieves close to optimal results based on a doubling dimension-based analysis in an inframetric model. Finally, extensive evaluation via simulation and PlanetLab experiments shows that MCRing is able to locate near-optimal relays.**

***Index Terms*— Relay communication, latency, concentric ring, inframetric, doubling dimension.**

I. Introduction

A. Motivation

ALTHOUGH the Internet was designed to enable pairwise communication among end hosts, direct communication may not be possible for various reasons: For example, many end hosts behind NATs or firewalls cannot directly reach each other. Also, detour routing [1]–[3], online multiplayer game [4], [5], VoIP [6], anonymous communication [7], outsourced middlebox processing [8] explicitly require a relay to act as an intermediate that forwards messages in real-time to clients. The forwarding latency must be as small as possible in order to meet the soft deadlines of the applications since a high latency may severely degrade the quality of experiences (QoE) of experienced by clients.

Manuscript received August 7, 2016; revised March 2, 2017 and June 6, 2017; accepted June 9, 2017; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor Y. Yi. Date of publication June 27, 2017; date of current version October 13, 2017. This work was supported by the National Natural Science Foundation of China under Grant 61402509. *(Corresponding author: Yongquan Fu.)*

Y. Fu is with the Science and Technology Laboratory of Parallel and Distributed Processing, College of Computer, National University of Defense Technology, Changsha 410073, China (e-mail: yongquanf@nudt.edu.cn).

E. Biersack, retired, was with Eurecom, Sophia Antipolis 06410, France (e-mail: erbi@e-biersack.eu).

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors. It provides a rigorous theoretical model based study of the effectiveness of locating optimal relays.

Digital Object Identifier 10.1109/TNET.2017.2715331

Network-Level Relay Communication: Detour routing [1], [2], [9]–[11] and outsourced middlebox processing [8] choose a relay on the direct path between a pair of communicating clients and forward real-time network packets close to the line speed. While network-level relay communication assumes a pair of clients, we envision that the same technique can be extended to a multicasting context that consists of more than two clients.

For example, [8] proposes to select servers from the Amazon cloudfront content distribution network (CDN) to detour packets that consists of 59 geo-distributed sites as of 2017 [12]. Akamai SureRoute [13] leverages over 170,000 edge servers in over 1,300 networks in 102 countries for detour routing [14], which probes the network latency between relays and between them and the clients, and then selects an optimized relay primarily based on the network latency towards a pair of clients.

Unfortunately, making centralized routing decisions does not scale well with increasing numbers of relays and clients, as relay communication requires on-demand delay probes to select the relay, instead of geographic distances for the following reasons: First, the geographic distances are static, however, wide-area delay values are dynamic due to changing load in the Internet. Second, the geographical distances assume the triangle inequality to hold. However, various empirical studies [1], [8], [15] have shown that the triangle inequality violations (TIV) are frequent. The TIV is defined as follows: Let d represent a pairwise distance function, if $d_{AC} + d_{CB} < d_{AB}$ holds for a triple (A, B, C), then an TIV arises. In fact, rerouting via a relay exploits the TIV to provide a better path. Therefore, choosing a suitable detouring relay that minimizes the performance overhead is the key for fast network-level relay communication.

Application-Level Relay Communication: Online multiplayer game [4], [5], VoIP [6], anonymous communication [7] offer multiparty-communication where a server explicitly forwards messages among a group of clients. For instance, VoIP applications use the relay to forward voice data between users and to bootstrap the communication between end hosts that are behind NATs or firewalls; online multiplayer game servers not only relay game updates to clients that join in the same group in order to keep the game states up-to-date for every player, but also filter out unwanted messages and broadcast notices. As humans are within the critical path of interaction, the delay from the relay to clients must be as low as possible since humans are acutely sensitive to delays.

Due to the large number of clients, service providers need to provision enough servers for handling group

sessions. In practice, these servers are placed in geo-distributed data centers. For example, the Amazon AWS cloud hosts over 20 geo-distributed regions, Microsoft Azure hosts 38 regions around the world. However, it is still an open problem on how to provision low-latency relays using geo-distributed servers for application-level multiparty communication. As each group is assigned to a fixed server, it is likely that the relay handling the traffic for a group may have large latency to clients in that group. This is may be because of large geographic distances or high queueing delays.

In summary, determining the relay with minimal network latency in a scalable way is challenging for both, network-level and application-level relay communication. Moreover, we need to be aware of the load status (e.g., CPU, memory, bandwidth) of the relays and avoid using overloaded servers as relays.

B. Existing Studies

A straightforward approach to optimal relay selection is to measure on demand the network latency between candidate relays and each client. However, this not only requires a centralized entity to collect all-pair network latencies but also consumes a lot of bandwidth resources. Furthermore, the set-up delay is considerable since the probing process needs to be completed before one can use the optimal relay.

Network coordinate methods embed nodes into low-dimensional synthetic coordinate systems and predict the pairwise RTTs using the pairwise coordinate distances. For example, [4] selects game players whose pairwise network coordinate distances are minimized based on the optimized Vivaldi network-coordinate algorithm [16]. Unfortunately, the network coordinate methods are still not widely adopted, since the pairwise RTTs are not strictly low-dimensional, the network coordinate system has a degree of latent prediction errors, which degrades the accuracy to minimize the latency between the relay and clients.

Meridian [17] and later improvements [18], [19] organize relays into an overlay that retains a relatively large number of nodes in vicinity, and uses on-demand measurements to recursively select a relay that is closer to clients. Unfortunately, we found that the optimal relay is generally outside of the immediate vicinity of the current node (see Subsection II-C.3). This is because as clients may be located arbitrarily in the Internet, the nearest relay with respect to a group of clients may be also arbitrary. However, when the current node is far away from clients, most of its neighbors are also far away from these clients, as each node favors nodes within its own immediate vicinity. As a result, the search process terminates at poor local minimum.

C. Our Work

In this paper, we present the design and implementation of a distributed system MCR that finds load-aware and latency-optimal relays for arbitrary combinations of geo-distributed clients.

First, we propose a novel overlay structure named maximum-coverage ring (**MCRing** for short) that addresses the limitations of existing approaches. Our key observation is that nodes in immediate vicinity in the latency space have similar distances towards other nodes. Therefore, in order to locate a closer relay, we need to be aware of the proximity clustering in the latency space. A naive approach would perform pairwise proximity clustering based on the all-pair network latency, which does not scale for large-scale systems. In order to scale the clustering process, as nodes from the same proximity region share similar distances towards other nodes, nodes from the same proximity region should have similar distances towards the current node. As a result, instead of pairwise proximity clustering, we only need to map network latency from other nodes to the current node into proximity clusters.

To that end, for each node P, we compute the proximity clusters of network latency from itself to other nodes based on K-means clustering and keep a modest number of relays in each proximity cluster. The clustering process relies on a gossiping process, which is lightweight and adapts well to dynamic relays.

Second, we propose a series of algorithms to accurately locate a relay that is closer to clients and satisfies the load constraints. Existing approaches fail to bound the relay-search performance with theoretical models that adapt to the triangle inequality violations in the network latency space [15]. We resolve this issue by proposing an analysis framework based on the doubling dimension that allows us to provide tight performance bounds of the relays found, which is available in the online supplemental material.

Finally, extensive experimental results and a PlanetLab deployment confirm that MCR is able to choose close-to-optimal relays for varying sizes of clients within 0.5 second and with modest bandwidth costs.

Going forward, Section II presents the problem model. Next, Section III introduces the overview of our approach. Section IV next presents MCRing management. Afterwards, Section V presents the process of selecting load-aware latency-optimal relays. Section VI presents the experiments using popular latency data sets and a real-world deployment on the PlanetLab. Finally, we conclude in Section VII.

II. Problem Statement

We next formulate the relay-search problem and identify the performance requirements. Then, we provide a realistic theoretical model to shed light on how to find good candidate relays. Finally, we summarize existing relay-search approaches and discuss their weaknesses to fulfill the performance requirements.

A. Problem Formulation

We next formulate a relay-search optimization framework. Table I lists key notations used in the paper.

Let S denote the whole set of nodes. Let S_T denote a specific group of clients that require the relay based message forwarding. Let S_r denote the set of relays. Let $S_{r_v} \subseteq S_r$ denote the set of relays that violate the load constraints. Let $\overline{S_{r_v}} \subseteq S_r$ be the complement set of S_{r_v} that satisfy the load constraints.

TABLE I
KEY NOTATIONS IN THE PAPER

Notation	Meaning
V	Set of all nodes
S_T	Set of clients
S_r	Set of relays
N	Number of relays
L	Number of clients in a request
d	Pairwise delays
$\gamma_\rho^{S_T}$	doubling dimension
Γ	K-means clustering partition
μ	K-means clustering centroids
β	Latency-reduction threshold
ρ	Inframetric parameter
C_c	Number of clustering rings
C_s	Number of vicinity rings

For a relay P from $\overline{S_{r_v}} \subseteq S_r$, an unbiased estimator of the expected delay amounts to the average latency from P to clients : $\frac{\sum_{j\in S_T} d_{Pj}}{|S_T|}$. Generally, we can quantify the responsiveness of each relay P with respect to clients S_T using a weighted latency:

$$\bar{d}_{PS_T} = \sum_{j\in S_T} w_{Pj} d_{Pj} \tag{1}$$

where w_{Pj} denotes the weight of the latency d_{Pj}. To represent the expected delay to deliver each message to all clients, we set $w_{Pj} = \frac{1}{|S_T|}$.

Our objective is to select a node P_* from the set $\overline{S_{r_v}}$ as the relay that has the minimal average latency to clients and does not violate the load constraints:

Definition 1:

$$P_* = \arg\min_{P\in\overline{S_{r_v}}} \bar{d}_{PS_T} \tag{2}$$

We identify two fundamental requirements for the relay-search approach:

- **Scalable**: As each relay's storage, computing and communication resources are limited, the relay-search process should minimize the overhead with increasing number of relays. Moreover, the overhead should be smoothly amortized among each relay.
- **Accurate**: The relay-search process needs to use the ground-truth latency to find the relay, such that the found relay optimizes forwarding latency as much as possible.

B. Understanding the Relay Search Problem Using the Inframetric Model

As the network latency space exhibits a degree of TIVs, we must consider a general theoretical framework that faithfully models the real characteristics of the Internet delay space.

Fraigniaud *et al.* [20] have proposed the inframetric model for triples of nodes that accounts for TIV to occur. The inframetric model is defined as follows: A distance function $d : V \times V \to \Re^+$ is a ρ-inframetric ($\rho > 1$), if d satisfies the following conditions for any pair of nodes u and v: (i) if $d(u, v) = 0$, then $u = v$; (ii) $d(u, v) = d(v, u)$; (iii) $d(u, v) \leq \rho \max\{d(u, w), d(w, v)\}$, for any arbitrary node w satisfying $w \notin \{u, v\}$.

The original inframetric model is defined for triples of nodes and requires the symmetry of pairwise RTTs, i.e. $d(P_1, P_2) = d(P_2, P_1)$. However, in the Internet, pairwise latency can be asymmetric [18], [21]. Therefore, we extend the inframetric model as follows:

Definition 2: A distance function d is called a (S_T, ρ)-inframetric (where $\rho \geq 1$), iff d satisfies the following conditions: (i) For any pair of nodes P_1 and P_2, where $P_1, P_2 \in V$, $d(P_1, P_2) = 0$, then $P_1 = P_2$; (ii) For a triple (P, Q, S_T), where $P, Q \in V$ and the relays in S_T

$$\begin{aligned} d_{PQ} &\leq \rho \min\left\{\max\left\{\overline{d}_{PS_T}, \overline{d}_{S_TQ}\right\}, \max\left\{\overline{d}_{PS_T}, \overline{d}_{QS_T}\right\}\right\} \\ \overline{d}_{PS_T} &\leq \rho \min\left\{\max\left\{d_{PQ}, \overline{d}_{QS_T}\right\}, \max\left\{d_{PQ}, \overline{d}_{S_TQ}\right\}\right\} \\ \overline{d}_{QS_T} &\leq \rho \min\left\{\max\left\{d_{QP}, \overline{d}_{PS_T}\right\}, \max\left\{d_{QP}, \overline{d}_{S_TP}\right\}\right\} \end{aligned} \tag{3}$$

hold.

We next bound the set of candidate relays that are closer to the relays S_T than the current node P using Lemma 3.1 in [19]. For a set S_r of nodes from the set V, let $B_P(r)$ be a **closed ball** centered at node P **covering** the set of nodes whose distances to node P are at most r:

$$B_P(r) = \{Q | d(P, Q) \leq r, P, Q \in S_r\} \tag{4}$$

where r denotes the **radius**. We have:

Lemma 3: Assume that there exists a node Q that is at least β ($\beta \leq 1$) times closer to relays, then node Q must be included in the closed ball $B_P(\rho\overline{d}_{PS_T})$.

We sketch the key ideas of the proof due to its importance for the relay search. Based on Eq (3), we see that

$$d_{PQ} \leq \rho \min\left\{\max\left\{r, \overline{d}_{S_TQ}\right\}, \max\left\{r, \overline{d}_{QS_T}\right\}\right\}$$

Since the minimum of a pair of numbers is never larger than any of these two numbers, we have

$$d_{PQ} \leq \rho \max\left\{r, \overline{d}_{QS_T}\right\}$$

Since we know that $\overline{d}_{QS_T} \leq \beta r$ and $\beta \leq 1$, we can see that $d_{PQ} \leq \rho r$. As a result, node Q is covered by the ball $B_P(\rho r) = \{x | d(P, x) \leq \rho r, P, x \in S_r\}$.

Lemma 3 shows that, in order to locate a node Q that is β times closer to the relays than the current node P, we need to scan the nodes within the closed ball $B_P(\rho d_{PS_T})$.

C. Analysis of Existing Approaches

Having formulated the requirements of the relay search problem, we next summarize existing approaches and discuss their weaknesses.

1) Exhaustive Approach: In order to obtain the most accurate relay, a straightforward approach is to collect all-pair latency values from each relay to the given group of clients. Then, we sort the latency values from each relay satisfying the load constraints to clients and select the relay with the smallest average latency towards clients. A special case is to select a relay for two clients. Detour [1], RON [2] and Nakao and Peterson [22] choose the optimal relay node to

forward packets between a pair of nodes. VIA [23] keeps a centralized-managed overlay that consists of stable servers from geo-distributed data centers, and selects relays based on historical measurements and network tomography with better scalability.

The exhaustive approach does not scale well with increasing numbers of relays and clients. For a single request, the overall probing cost is $O(N \cdot |S_T|)$ for N relays and $|S_T|$ clients, and the computing complexity is $O(N \log N)$ based on the heap-sort method. Further, the exhaustive approach needs to frequently update the latency measurement from all relays to clients, since clients may join the system at will and the wide-area network latency varies dynamically.

2) Greedy Approach: A more scalable approach is to recursively choose the relay that has a smaller average latency to clients than the current node, which avoids the performance bottleneck of the exhaustive approach by allowing any relay to answer the relay-search requests.

Definition 4 (Greedy Relay Search): Each relay maintains a set of online relays (called ***neighbors****). A greedy relay-search process starts at a random relay P; node P tries to select a relay Q from its neighbor set that is β times closer to clients than node P:*

$$\bar{d}_{Q\mathbf{T}} < \beta \bar{d}_{P\mathbf{T}} \tag{5}$$

where $\beta \in (0, 1]$. If such a relay Q exists, then node Q recursively selects a relay that is β times closer to clients than $\bar{d}_{P\mathbf{T}}$. The search terminates when no such a node Q can be found.

PeerWise [3] and IRS [24] relay for a pair of nodes that have a high embedding error, hoping to exploit the TIV to reduce the end-to-end delay via the relaying path. While the embedding error is correlated with the TIV [3], it is caused by a combination of factors such as changing network distances, convergence of the embedding algorithm, coordinate drifting [25]. As a result, the relaying performance lacks guarantee.

Based on the discussions of the greedy approach, we can see that there is a fundamental trade-off between the accuracy and the maintenance overhead. We call a maintained set of relays *compact* if the number of relays is at most a polylogarithmic function of the total number of relays. A compact relay set scales well with an increasing number of relays.

Although the greedy approach scales well, guaranteeing the optimality of the relay found is challenging due to the missing of global information, which leads to the problem we tackle in this paper.

3) Meridian Approach: Meridian is a greedy approach. For a request, Meridian locates a next-hop node that is β times closer to clients than the current node. A Meridian node P measures its delay $\overline{d}_{PS_T}$ to the targets S_T, then node P selects candidate neighbors whose delays to P are within $[(1-\beta)\overline{d}_{PS_T}, (1+\beta)\overline{d}_{PS_T}]$.

The key component of Meridian is a compact data structure called **concentric ring** that is centered at each node P. A concentric ring keeps a number C of rings, where neighbors are put in the i-th ring if their latency values towards node P are within the interval $\left(\alpha s^{i-1}, \alpha s^{i}\right]$, with $i > 0$, α a constant, s a multiplicative increase factor ($\alpha = 1$, $s = 2$ ms by default). Each ring keeps a fixed number of neighbors on the order of $O(\log N)$ for N relays. The inner-most and outer-most rings also collapse potential neighbors whose latency values to node P are smaller than α and greater than αs^{C}, respectively.

To fill nodes into a concentric ring, each node periodically finds new neighbors via an anti-entropy gossip protocol. Meridian selects a subset of found nodes into the concentric ring by maximizing their geographic diversity via a maximum hypervolume polytope algorithm [17].

Unfortunately, our experiments show that Meridian may be trapped at the local minimum [18], [26], which naturally raises the question whether the exponential rule to organize rings is necessary for the relay search process. To that end, we quantify the relation between the distance from the current node to the optimal relay and the accuracy of the found relay.

(i) *Experimental methodology*: We have built a simulator whose details are given in Subsection VI-A.3. We sample a set of relays and a group of clients from real-world RTT data sets (details introduced in Section VI-A.1), we configure each relay's concentric ring as sixteen nodes per ring, nine rings per node, $\alpha = 1$, $s = 2$. We set the termination threshold β to one. We next greedily search the best relay that is closest to clients via Meridian; meanwhile, we globally determine the optimal relay that has the shortest average distance to clients.

We measure the accuracy of the found relay using the **ratio** between the average distance from the found relay to clients and that from the globally optimal relay to clients. We can see that the ratio is greater than or equal to one, and the smaller the ratio, the more accurate the found relay.

(ii) *Results*: From Figure 1 we can see that when the search terminates, the RTT values from the current node to the optimal relay vary by over six orders of magnitude. Consequently, the optimal relay is generally outside of the immediate vicinity of the current node.

The greedy relay-search process inherently centers at clients and requires to choose a relay in the immediate vicinity of clients. As clients are located arbitrarily, the nearest relays vary arbitrarily as well. For the concentric ring approach, each node favors nodes within its own immediate vicinity and when the current node is far away from clients, most of its neighbors are also far away from these clients. Therefore, favoring the immediate vicinity is not conclusive.

III. Overview

In this paper, we present a novel overlay MCRing for scalable and accurate relay search. We present detailed MCRing in the next section and the relay-search algorithms in Section V.

A. Key Observation

As geo-distributed clients are unknown *a priori*, the relays that are within the "immediate vicinity" of clients may be arbitrary in the latency space. As a result, selecting a relay close to clients needs to be aware of the proximity structure of the latency space.

Our key insight is that, the wide-area latency space exhibits proximity regions, where nodes in the same proximity region typically have similar distances towards other nodes, as shown

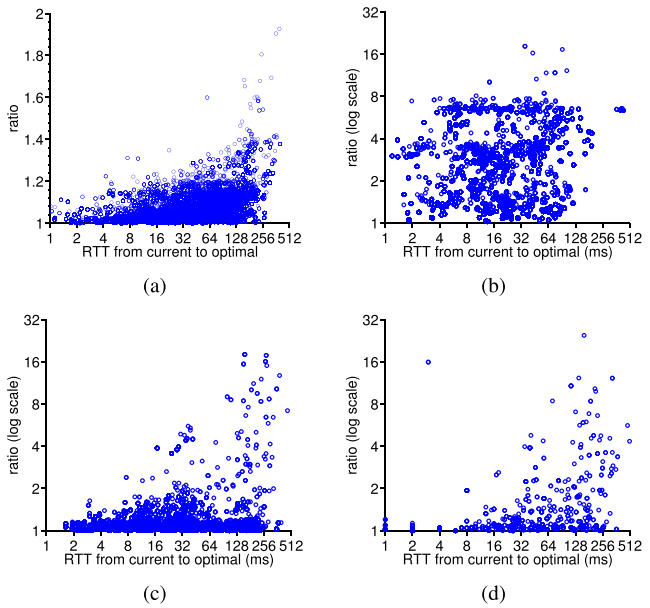


Fig. 1. The ratio of the average distance from the found relay to clients and that from the optimal relay to clients VS. the RTT from the current node to the optimal relay for Meridian. (a) P2P1143. (b) M2500. (c) King3997. (d) end479.

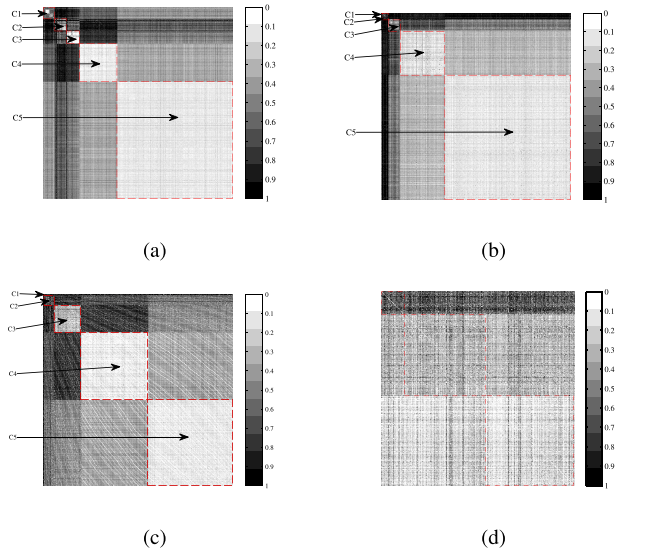


Fig. 2. Clustering structure of the delay space. The data sets are introduced in the simulation section. We estimate the clustering structure of the network delay space based on (a) P2P1143. (b) M2500. (c) King3997. (d) end479. [27].

in Figure 2. As a result, in order to locate a nearby relay for arbitrary combinations of clients, each node needs to sample neighbors from each clustering region in the latency space.

B. Architecture

We propose a distributed relay-search system MCR. Figure 3 summarizes major components.

The key substrate is a novel proximity-cluster-aware overlay MCRing. It maintains a compact set of neighbors from proximity clusters in the latency space. To maximize the coverage

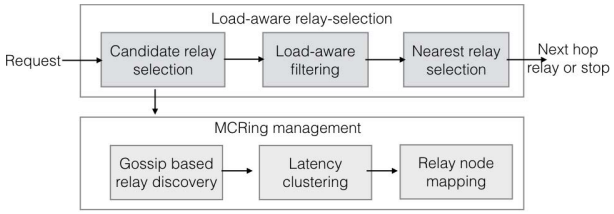


Fig. 3. relay-search components.

of the cluster structure in the latency space, a naive approach would find the proximity clusters based on all-pair latency values among relays, which however, does not scale well for large-scale and dynamic systems. Our key insight is that, for each relay P, nodes from the same cluster should share similar distances to node P, therefore, we can partition nodes to proximity clustering according to their latency values towards the current node P.

To tolerate the dynamics of the relay and to avoid the single point of failure, we propose a gossiping based K-means clustering method to discover the clustering structure in a decentralized manner. Each node P independently computes the proximity clusters during the gossiping process, and keeps a modest number of neighbors from each of these proximity clusters.

We next propose a load-aware distributed greedy relay-search process. Each node P seeks a relay that simultaneously fulfills the load constraint in the message and is at least β ($\beta \in (0, 1]$) times closer to clients. The search terminates when no such a better relay can be found. In order not to miss relays that may be closer to clients, we present a series of algorithms to locate a relay and establish rigorous performance guarantees.

C. MCRing Structure

Based on the proximity clustering, MCRing maintains two classes of rings:

- **Vicinity rings**: We maintain C_s ($C_s = 2$ by default) rings with $(0, 8]$ and $(8, 16]$ as two latency intervals that are derived from real-world RTT data sets in Subsection VI-A.1. For each relay Q, if the latency d_{PQ} falls within a latency interval of a vicinity ring, we map relay Q to this ring.
- **Clustering rings**: We partition found relays to C_c proximity clusters. For each relay Q that is not included in the vicinity ring, we map this relay to the ring whose clustering centroid is closest to the latency d_{PQ}.

Our analysis shows that, we only need $O(\log(N))$ neighbors in the MCRing in order to find approximately optimal relays.

We next present an example of a MCRing in Figure 4. We randomly generate a number of latency samples from a node P to other nodes and construct a MCRing for a node P. There are five rings, where two innermost rings use $(0, 8]$ and $(8, 16]$ as their latency intervals, respectively. For each relay Q that is not mapped to the vicinity rings, we map node Q to the ring whose clustering centroid is closest to the latency d_{PQ} from node P to node Q. The other three rings calculate the

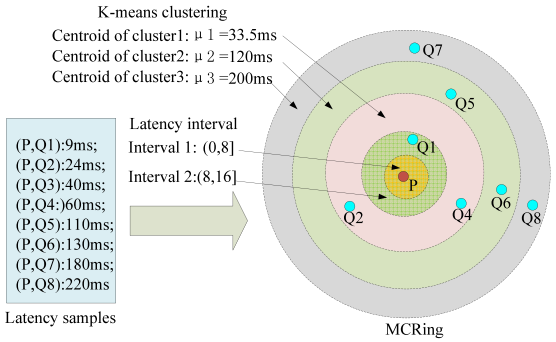


Fig. 4. An example to construct the MCRing.

clustering centroids based on Algorithm 1, which are 33.25, 120 and 200, respectively. We assign the clustering centroids to these rings in an ascending order.

We next map relays to the MCRing: we map Q1 to the second ring since $d_{PQ1} \in (8, 16]$, Q2 and Q4 to the third ring since they are closest to the centroid 33.25, Q3 to be removed since each ring has at most two relays, Q5 and Q6 to the fourth ring since they are closest to the centroid 120, and Q7 and Q8 to the fifth ring since they are closest to the centroid 200.

We can see that the MCRing preserves typical latency values from node P to other nodes, while for a concentric ring approach with the latency intervals as $\left(2^{i-1}, 2^i\right]$ where i denotes the index of the ring. The latency intervals of five innermost rings are $(0, 2]$, $(2, 4]$, $(4, 8]$, $(8, 16]$, $(16, +\infty]$, respectively. We can see that Q1 is mapped to the fourth ring, while the other nodes are mapped to the fifth ring. As we only sample at most two relays for each ring, we can see that the concentric ring fails to cover the typical clusters of the latency values from node P to other nodes.

IV. MCRing Management

Having presented the overview of the relay-search system, we next introduce the overlay MCRing that summarizes the proximity clusters of the latency space in a compact way.

A. MCRing

1) Proximity Clustering: Given a vector of latency values from a set S_P of sampled relays to node P, we map this vector into a number of clusters, where each cluster consists of nodes that have similar distances to node P. Partitioning a vector of latency values into clusters is essentially a vector quantization problem: keeping optimized clusters of relays is equivalent to a vector quantization process of the latency values from the discovered neighbors to the current node. As a result, we reduce the all-pair clustering problem into a "vector quantization" problem, which has been extensively studied in signal processing and machine learning fields. In order to scale well and adapt to dynamic relays, the clustering process should be lightweight and maintain the dynamic proximity clusters. We select the well-known K-means clustering method [28] for its simplicity and good performance. Other vector-quantization methods may slightly improve the performance, but the same conclusions still hold.

Let C_c denote the number of clusters. Let $\Gamma = \{\Gamma_1, \Gamma_2, \ldots, \Gamma_{C_c}\}$ represent a partition of the latency samples from node P to its relays. The K-means clustering method seeks to minimize the distances of each node in that cluster with respect to this cluster's centroid, where the **centroid** μ_l of the l-th group $(1 \le l \le C_c)$ amounts to:

$$\mu_l = \frac{\sum\limits_{x \in \Gamma_l} x}{|\Gamma_l|} \tag{6}$$

The clustering quality function F can be stated as follows:

$$F(\Gamma) = \sum_{j \in S_P} \left\| d_{Pj} - \mu\left(d_{Pj}\right) \right\|^2 \tag{7}$$

where $\mu\left(d_{Pj}\right)$ denotes the centroid of the group to which d_{Pj} belongs.

2) Immediate Vicinity: Moreover, we explicitly keep a small number of neighbors from the immediate vicinity to complement the K-means clustering that captures a macroscopic-level structure. When the current node P is close to clients, node P needs to select a next-hop relay from its own immediate vicinity.

B. MCRing Maintenance

We maintain the vicinity rings and clustering rings based on a lightweight gossiping based process.

1) Gossiping Based Relay Discovery: We discover neighbors based on a lightweight gossiping process. When a node P joins the system, node P obtains a number of online nodes as its initial relays from a bootstrapping server. Then, node P probes latencies to these relays, and stores those with successful responses to a temporary list.

To be aware of load conditions of neighbors in the MCRing, each node piggybacks its load status to its neighbors through the gossip messages during the MCRing maintenance period.

Further, we simultaneously perform the K-means clustering within the gossip process. We optimize the clustering centroids with the Lloyd method [28] that adapts well to dynamic samples. We initialize the centroids to be random values and iteratively adjust the centroids to stabilized positions based on Algorithm 1. The K-means clustering method incrementally adjusts its clustering structure in order to find a partition Γ that minimizes the objective:

$$\Gamma_{opt} = \arg\min_{\Gamma} F(\Gamma) \tag{8}$$

Algorithm 2 summarizes the gossip process between two online nodes. The probing cost is amortized using the gossip process where a node contacts one relay and probes the latency to one relay in one round. Moreover, the K-means clustering does not incur additional bandwidth cost.

2) Relay Node Mapping: Along with the K-means clustering process, we dynamically maintain the mapping between sampled neighbors and the corresponding ring. Line two in Algorithm 1 summarizes the mapping logic. We assign the clustering centroids to C_c rings in an ascending order, where the i-th ring is assigned the i-th smallest clustering centroid. Then, for each neighbor Q that is not included in the vicinity ring, node P maps this relay to the ring whose clustering centroid is closest to the latency d_{PQ} based on Eq (9).

Algorithm 1: Latency Clustering

`LLoydCluster(`P`,`d_{PQ}`,`μ`)`

input : Node P, the latency sample d_{PQ}, the clustering centroids μ.
output: The centroids μ.

Cluster update. Node P maps the latency sample d_{PQ} to the closest centroid:

$$s = \arg\min_{\Gamma_s \in \Gamma} |d_{PQ} - \mu_s| \tag{9}$$

Centroid update. Node P updates the s-th clustering centroid based on Eq (6);

Algorithm 2: Gossip Based Relay Sampling and Latency Probe

`GossipProbe(`P`)`
input : A temporary list that consists of bootstrapping nodes.
while *TRUE* **do**
  Node P selects a relay Q uniformly at random from the union of its own MCRing and the temporary list;
  Node P sends node Q a request message consisting of a randomly sampled relay R_P from its MCRing at time $T0_Q$;
  if *Node Q receives the request message from node P* **then**
    Node Q stores the relay R_P into its the temporary list ;
    Node Q sends P a response message comprising a randomly sampled node R_Q from Q's own MCRing;
  if *Node P obtains the response message from node Q at $T1_Q$* **then**
    Node P updates the latency to node Q using $d_{PQ} = T1_Q - T0_Q$;
    $\mu =$ `LLoydCluster(`P`,`d_{PQ}`,`μ`)`;
    Node P probes the latency to the node R_Q;
    if *Probe to the node R_Q succeeds* **then**
      if $d_{PR_Q} \in (0, 16]$ **then**
        Node P puts node R_Q to the corresponding vicinity ring;
      else
        Node P saves node R_Q to the temporary list;
        $\mu =$ `LLoydCluster(`P`,`d_{PR_Q}`,`μ`)`;
  Sleep t seconds;

3) Ring Replacement: To ensure a compact MCRing, we set up a threshold to bound the maximum number of relays each ring, which is a polylogarithmic function of the number of relays. If the number of sampled relays in the MCRing exceeds the threshold, we move a number of relays uniformly at random into the temporary list that is not used for the relay-search process.

Our experiments show that the random replacement of neighbors leads to good performance. While [17] seeks to keep relays that maximize pairwise distances among themselves, however, we found that this sophisticated approach [17] not only consumes additional probing overhead, but also is sensitive to nodes that are far away from most of nodes, as these nodes are useless for most relay-search requests.

4) Trade-Off: In order to balance the accuracy and the scalability of the MCRing, several factors need to be balanced:

- **Number of Rings**: Increasing the number of clusters decomposes the latency distribution to more fine-grained clusters, however, the bandwidth cost also increases with more rings.
- **Number of relays in Each Ring**: Increasing the number of relays in each ring improves the coverage in a cluster, but also increases the bandwidth cost.

V. Relay Search

Having presented the MCRing structure, we next propose the relay-search process in details.

A. Workflow

When a relay-search request is sent to the system, the first relay (denoted as the requestor) that receives the request initializes a greedy relay-search process. After the greedy process successfully completes, the requestor obtains the address of the found relay and returns to the upper-layer application.

The requestor builds a relay-search message that is comprised of the requestor' address, the set of clients' addresses S_T, a load threshold τ to filter overloaded relays, and the addresses S_r of traversed relays to avoid self-loops: *If a candidate relay has appeared in S_r, i.e., it must have forwarded the same message and should be filtered out*.

For each relay P that receives the relay-search message, it performs the following steps:

1) Node P probes clients and computes the average latency r.
2) Node P selects candidate relays that may be closer to clients from its own MCRing (Subsection V-B).
3) Node P locates a relay Q (node Q may be node P itself) with the smallest average latency to clients (Subsection V-C).
4) If node Q is at least β times closer to clients than node P, node P forwards the relay-search message to node Q; otherwise, node P terminates the search process and sends node Q's address to the requestor (Subsection V-D).

B. Candidate Relay Search

First, node P probes the average RTT r towards clients that are embedded in the request message. Then, node P iterates from the innermost ring to outer rings in order to choose the candidate relays. We show that (see Lemma 3), the latency values from node P to candidate relays should be not larger than ρr, where ρ denotes a global parameter of the latency space, $\rho = 3$ by default.

Algorithm 3: Select Candidates From the MCRing

```
1 Candidate(P, r, msg)
        input : Current node P, average distance from P
                to clients r, the relay-search message msg.
  output: S_{L_c}.
2 S_{L_c} = ∅ ;
3 for i = 1 → C_s + C_c do
4     if ρr < min_{{Q_i|Q_i∈ring−i}} d_{PQ_i} then
5         Break;
6     else if ρr ≤ max_{{Q_i|Q_i∈ring−i}} d_{PQ_i} then
7         S_{L_c} = S_{L_c} ∪ {Q_i|d_{PQ_i} ≤ ρr, Q_i ∈ ring − i} ;
8     else
9         S_{L_c} = S_{L_c} ∪ {Q_i|Q_i ∈ ring − i} ;
10 return S_{L_c};
```

Algorithm 4: Determine the Closest Candidate to Clients

```
1 Closest(P, S_{L_c}, msg)
        input : Current node P, candidates S_{L_c}, the
                relay-search message msg.
  output: The node A that is closer to msg.S_T and corre-
          sponding average distance d̄_{AS_T} to clients.
2 if S_{L_c} ≠ ∅ then
3     Node P filters out relays from S_{L_c} that have been
      included in msg.S_r or violate the load constraint msg.τ;
4     Node A ← online relay with the smallest average
      latency within S_{L_c};
5     return (A,d̄_{AS_T}) as the node closest to clients msg.S_T;
6 else
7     return (P,r) as the node closest to clients msg.S_T;
```

Algorithm 3 summarizes the steps of selecting candidates from a MCRing. As the rings are organized in accordance with increasing centroids of K-means clusters, we can see that if the maximum RTT from the relays in a ring to node P is smaller than ρr, then we select all relays in this ring as the candidates; if the minimum RTT from the relays in a ring to node P is greater than ρr, then we terminate the iteration process, since no relays in this ring or more outer rings satisfy the selection condition.

C. Load-Aware Filtering and Nearest Relay Search

After successfully locating candidate relays, we next determine the optimal candidate. Algorithm 4 shows the steps. We filter out candidate relays that either violate the load constraint, or have been recorded in the list S_r of traversed relays. Next, we select the relay with the minimum average RTT towards clients. Further, if no such candidate relays are available, we directly skip the above process and set the current node as the optimal relay.

Algorithm 5: Recursive Relay Search

```
1 Search(P, r, A, d̄_{AS_T}, r, msg)
        input : Current node P, average distance from
                P to clients r, selected relay A, average distance
                from A to clients d̄_{AS_T}, average distance from P
                to clients r, the relay-search message msg.
  output: The next-hop node that is closer to msg.S_T.
2 if d̄_{AS_T} ≤ βr then
3     msg.S_r = msg.S_r ∪ A;
4     Search(A, msg);
5     return ;
6 return (A,d̄_{AS_T}) to msg.requestor;
```

Algorithm 6: Load-Aware Relay-Search Algorithm

```
1 Search(P, r, msg)
        input : Current node P, average distance from P
                to clients r, the relay-search message msg.
  output: The next-hop node that is closer to msg.S_T.
2 S_{L_c} = Candidate(P, r, msg);
3 (A,d̄_{AS_T}) = Closest(P, S_{L_c}, msg);
4 (A,d̄_{AS_T}) = Search(P, r, A, d̄_{AS_T}, r, msg);
```

D. When to Stop

Afterwards, we determine whether we continue the recursive search. Algorithm 5 summarizes the steps. The current node forwards the message to the next-hop neighbor if we locate a relay that is at least β times closer than the current node P, otherwise, the search terminates and we return the optimal relay to the requestor.

E. Putting It All Together

Algorithm 6 summarizes the relay-search process. Line two selects candidates from the MCRing whose loads may violate the constraints (see Algorithm 3). Line three determines the neighbor that satisfies the load constraint and is closest to clients (see Algorithm 4). Line four decides whether to forward the message to a next-hop neighbor A or to return the currently optimal relay (see Algorithm 5).

VI. Performance Evaluation

We next perform experiments to show the performance of choosing the optimized relays.

A. Simulation Experiments

1) Data Sets for Simulation: We use four real-world data sets in this paper: (i) **P2P1143**, a symmetric RTT matrix between 1143 DNS servers by the MIT P2PSim project [29] with the King method [30]. (ii) **M2500**, a symmetric RTT matrix between 2500 DNS servers collected by the Meridian project [17] also with the King method [30]. (iii) **King3997**. A symmetric delay matrix collected between 3997 DNS name servers by Zhang *et al.* [31] using the King method.

TABLE II
THE CHARACTERISTICS OF THE PAIRWISE RTTS IN THE DATA SETS

Data	Mean (ms)	STD (ms)	Skewness
P2P1143	143.6	99.2	1.19
M2500	75.7	71.3	5.02
King3997	165.7	106.1	3.10
end479	150.9	138	1.91

(iv) **end479**. An asymmetric delay matrix based on aggregated delays from end hosts participating in BitTorrent [32].

We summarize the central tendency, the dispersion and the shape of all-pair RTT values based on the mean, standard deviation and the skewness metric that is defined as $\frac{E(x-\mu)^3}{\sigma^3}$ [33], where μ denotes the mean of all-pair RTTs, μ denotes the standard deviation of all-pair RTTs, $E(t)$ denotes the expected value of t. The results are shown in Table II.

We can see that different data sets have varying mean and standard deviation numbers. Further, the skewness is positive, indicating that the right tail of the all-pair RTT distribution is longer than the left portion, while the mass of the distribution is concentrated on the left portion. Moreover, the skewness varies across data sets due to the variation of the mean and the standard deviation.

2) Comparison of the Different Methods: We compare MCR's performance with the following methods:

- **Meridian**: maintains the concentric ring via the maximal hypervolume polytope algorithm and selects candidate relays from the rings numbered $\left[\log\left\lceil(1-\beta)\bar{d}_{PT}\right\rceil, \log\left\lceil(1+\beta)\bar{d}_{PT}\right\rceil\right]$. We set β to one in order to maximize the range of candidate relays.
- **CR-infra**: a modified Meridian approach based on the inframetric model. It uses the concentric ring to organize relays, but searches relays closer to clients based on the load-aware greedy method. In each step, it selects candidate relays from the rings numbered $\left[0, \log\left\lceil\rho\bar{d}_{PT}\right\rceil\right]$ of a node's concentric ring. We set ρ to three based on the measurements on the latency data sets.
- **Htrae-**: predicts the pairwise latency based on [4] and estimate the relay closest to clients based on the all-pair estimated latencies. We implement the TIV avoidance and history and the symmetric updates. But we are unable to perform the geographic bootstrapping and the AS correction due to the lack of domain knowledge. We set the number of relays to 32 for each node. We update the coordinates for each node in 50 rounds and then collect the pairwise coordinate distances.

We set MCR's default parameters to trade off between the accuracy and the maintenance costs according to the sensitivity analysis in Subsection VI-A.6: We set the number of rings to ten, the maximal number of relays per ring to eight, the inframetric parameter ρ to three, and the search threshold β to one.

For a fair comparison of Meridian and CR-infra, we use the same gossip process to obtain the relays and choose the same number of rings, the same number of relays per ring and the same search threshold β.

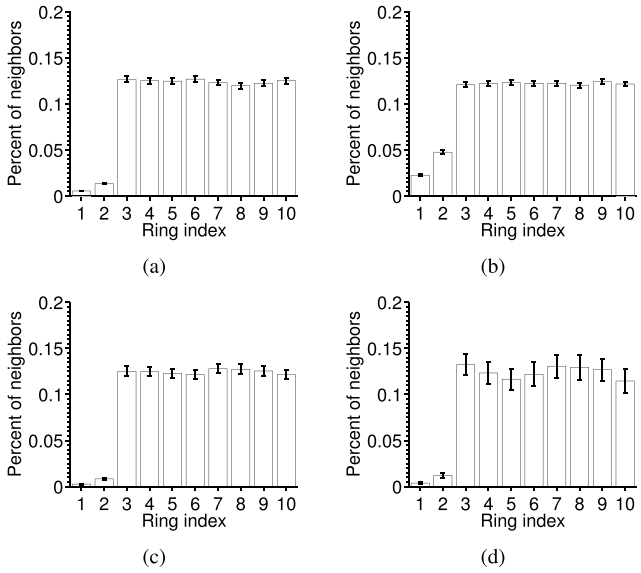


Fig. 5. The distribution of the fraction of relays mapped to each ring for MCRing. (a) P2P1143. (b) M2500. (c) King3997. (d) end479.

We compute the **absolute error** to quantify the difference of the average latency to clients between the found relay and the optimal relay, which is defined as follows:

$$\frac{1}{L}\left|\sum_{j=1}^{L} d_{P^*T_j} - \sum_{j=1}^{L} d_{\text{OptimalRelay}T_j}\right| \tag{10}$$

where L denotes the number of clients and $(T_1, \ldots, T_L)$ represents the set of clients, $P^* \in S_C$ denotes the found relay using the relay-search algorithm, OptimalRelay $\in S_C$ represents the relay that has the minimal average latency to clients.

3) Simulation Methodology: We have implemented a simulator for relay search. The simulator initializes the relays at the start of the simulation and performs event-driven simulations by generating and processing the relay-search requests. Each request concerns a set of clients sampled from the whole set of nodes. A request is randomly delivered to a candidate relay that initializes the relay-search process. The simulator sets up a warm-up period of 1,000 seconds. During the warm-up period, no relay requests are generated and each relay only maintains its neighbors.

In order to obtain the expected performance, we randomly sample a different set of clients for every four request, and each request is assigned to a randomly sampled relay. The simulation experiments consist of 10,000 synthetic requests.

4) MCRing Analysis: We report the percents of relays mapped to each ring in the MCRing. We set the total number of rings to ten. We configure two innermost rings with the latency intervals $(0, 8]$ and $(8, 16]$, respectively. We set the number of the clustering rings to eight. Then, we construct the MCRing for each node.

From Figure 5, we can see that except for two innermost rings that deliberately store relays with small latencies, the other eight rings have balanced numbers of relays. The uniform distributions of relays are primarily due to the K-means

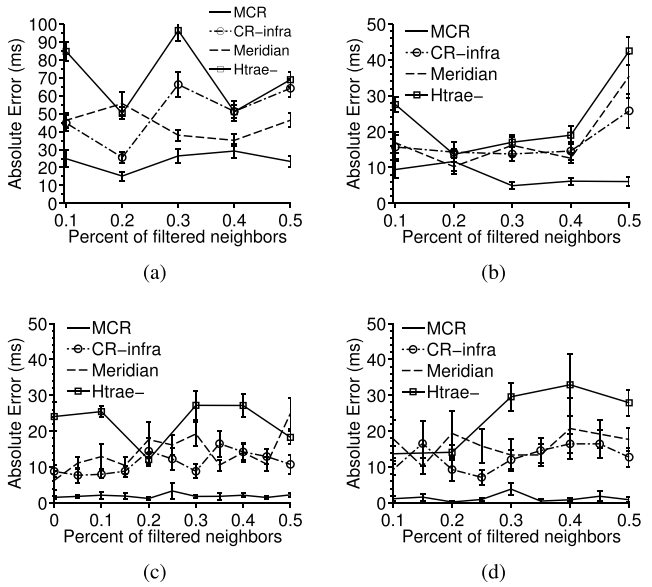


Fig. 6. Comparison of the mean absolute errors as a function of the percent of relays that are filtered due to overloads. (a) P2P1143. (b) M2500. (c) King3997. (d) end479.

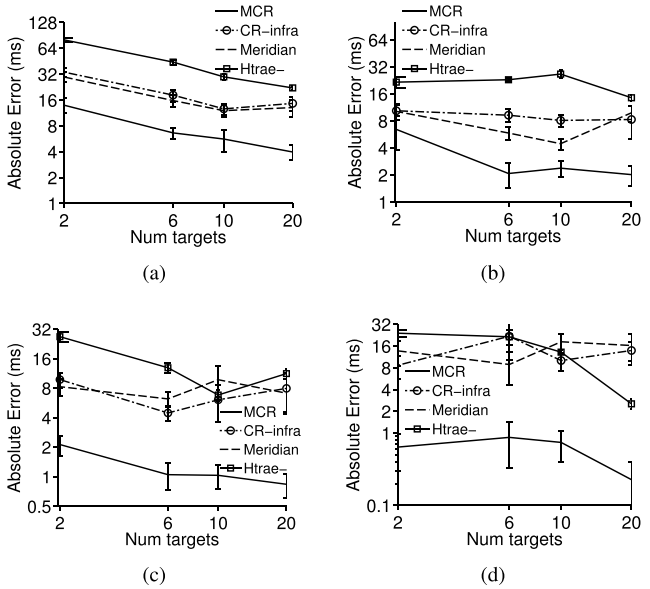


Fig. 7. The mean absolute errors as we vary the number of clients. (a) P2P1143. (b) M2500. (c) King3997. (d) end479.

clustering that maximizes the similarity of the latency values in the same group. As a result, we can randomly sample relays and obtain balanced rings.

5) Comparison: We next compare the performance of MCR with other relay-search methods.

(i) **Load-aware Search**

We first test whether the load-aware filtering yields latency-optimized relays. We select a fraction of relays of each node to be filtered during the relay-search process. Then we compute the absolute error between the found relay and the optimal one.

We set the number of clients to two, the number of rings to ten, the number of relays per ring to ten, the inframetric ρ to three, the threshold β to one.

Most search processes complete in two to three hops for MCR, CR-infra and Meridian. As a result, the relay can be found quickly. Therefore, our comparison focuses on the accuracy and the stability of the search processes.

Figure 6 shows the mean absolute errors and their 95-th confidence intervals. We can see that MCR is able to find near optimal relays with increasing numbers of filtered relays, thanks to the fine-grained coverage in the latency space. While CR-infra and Meridian increases the absolute errors with increasing filtered relays due to the poor coverage by the concentric ring. Further, Htrae- has the largest absolute errors compared to the other three methods. This is because we use the average latency to determine the closeness between the relay and clients, which amplifies the prediction errors in the network coordinate system.

(ii) **Varying the number of clients**

Having shown that MCR out-performs CR-infra and Meridian under filtered nodes, we next set the percent of filtered relays to zero in order to obtain the best absolute errors for CR-infra and Meridian. We next vary the number of clients and study the dynamics of the mean absolute errors.

As shown in Figure 7, MCR has the lowest absolute errors with increasing clients, as a result, MCR is robust against the choice of clients. Further, the absolute errors of all methods decrease as we add more clients. This is because we randomly sample clients, the average latency from the current node to clients converges to the global average latency value with increasing numbers of random samples.

(iii) **Comparing with modified concentric ring**

Having confirmed that MCR outperforms Meridian and CR-infra under the same parameter configuration, we next vary the configuration of the concentric ring such that it keeps the same total number of neighbors as in MCR, but does not fix the numbers of neighbors per ring.

For the modified concentric ring, first, we put as many neighbors as possible in each ring via a random sampling procedure; second, when the total number of neighbors in the concentric ring exceeds MCR's neighbor-set size, we iteratively remove neighbors from rings that have the largest numbers of neighbors until the number of remaining neighbors amounts to MCR's neighbor-set size. Meridian and CR-infra via the modified concentric ring are denoted as Meridian+ and CR-infra+, respectively.

We compare the absolute errors of relays found by Meridian+, CR-infra+ with those found by the original Meridian, CR-infra and MCR. We set the number of clients to two, the number of rings to ten, the number of relays per ring to ten, the inframetric ρ to three, the threshold β to one.

First, we plot the distributions of the numbers of probes required for a relay-search process. Figure 8 shows the complementary cumulative distribution functions (CCDFs) of the numbers of probes. We can see that Meridian+ and CR-infra+ probe much more neighbors than Meridian and CR-infra, since most of neighbors in the modified concentric ring are not included into the original concentric ring. Also, MCR probes fewer neighbors than Meridian+ and CR-infra, as MCRing

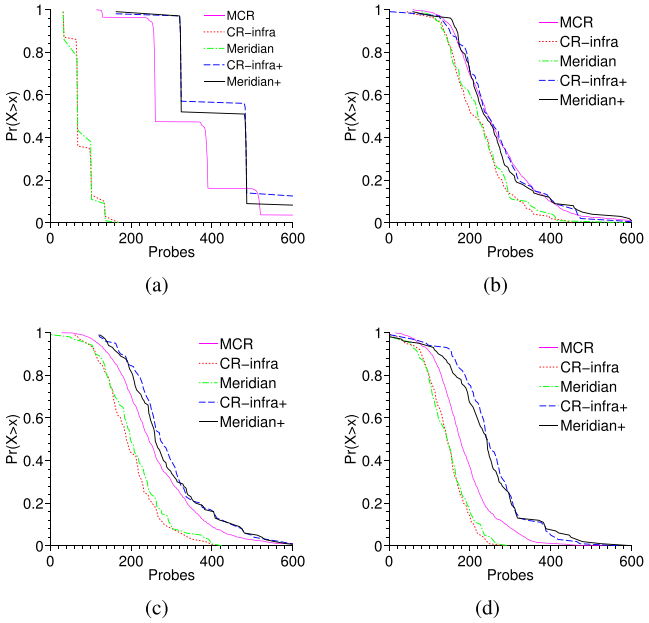


Fig. 8. Comparing the CCDFs of numbers of probes for MCR, the modified concentric ring based Meridian+ and CR-infra+, and the original concentric ring based Meridian and CR-infra. (a) P2P1143. (b) M2500. (c) King3997. (d) end479.

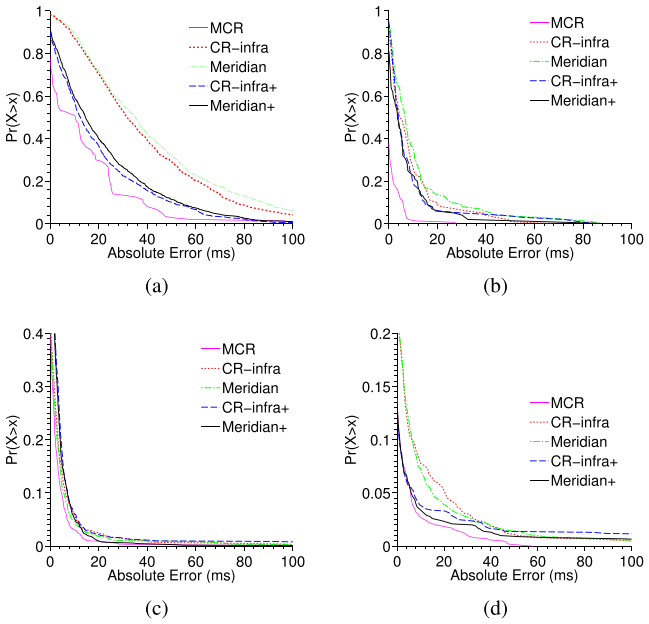


Fig. 9. Comparing the CCDFs of absolute errors of modified concentric ring based Meridian+ and CR-infra+ and the original concentric ring based Meridian and CR-infra, as well as MCR. (a) P2P1143. (b) M2500. (c) King3997. (d) end479.

enforces a bounded size for each ring, while the modified concentric ring approach relaxes the upper bound.

Second, we compare the absolute errors of the relay-search process. Figure 9 presents the CCDFs of the absolute errors of found relays. Meridian+ and CR-infra+ outperform Meridian and CR-infra, since most neighbors are located in quite a few rings, while the modified concentric ring is able to sample

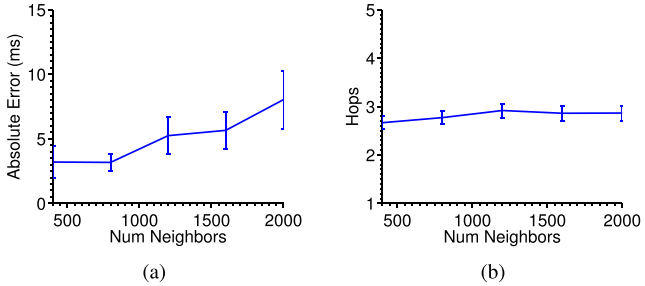


Fig. 10. Absolute errors as we increase the number of relays. (a) Absolute error. (b) Search hops.

more neighbors into corresponding rings than the original concentric ring.

We can see that more neighbors do not necessarily translate into better results. Meridian+ and CR-infra+ probe more neighbors than MCR (c.f. Figure 8), however, MCR has a much shorter tail of absolute errors than Meridian+ and CR-infra+. This is because MCRing directly samples neighbors from proximity clusters in the latency space, while the concentric ring is agnostic of the proximity clusters. As a result, MCRing based greedy relay search process has a higher probability of locating a relay within the immediate proximity of clients than the concentric ring based relay search process.

6) Parameter Sensitivity: Having shown that MCR obtains the lowest absolute errors compared to existing methods on four data sets, we next study MCR's performance as we change its parameter configuration. We report results on the M2500 data set. The same conclusions hold for the other three data sets.

(i) **Scalability**

We study the accuracy as we increase the number of relays in the system. From Figure 10 (a), we see that the average absolute errors increase marginally from four ms to about eight ms as we increase the number of relays from 400 to 2,000, since although the MCRing based greedy search process guarantees the approximate performance, more local minimum arise as we increase the number of candidate relays. Further, Figure 10 (b) shows that the search hops keep around three, which is consistent with the theoretical search-path length. As a result, the search can terminates fast with increasing numbers of relays.

(ii) **Number of Rings**

We next study the variation of the accuracy as we vary the number of rings. From Figure 11, we see that increasing the number of rings decreases the absolute errors of the found relay, since more relays are stored in the MCRing, which increases the coverage of the latency distribution. Further, six to ten rings are sufficient, since there are a small number of significant clusters in the latency distribution.

(iii) **Number of relays per ring**

Having shown that we can select a modest number of rings to obtain close-to-optimal relays, we now study the effect of the number of relays per ring on the search performance.

We vary the number of relays from four to twelve and compute the absolute errors. From Figure 12, we can see that

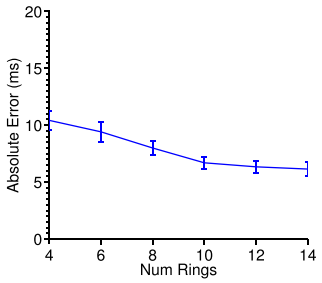


Fig. 11. Absolute error as we vary the number of rings.

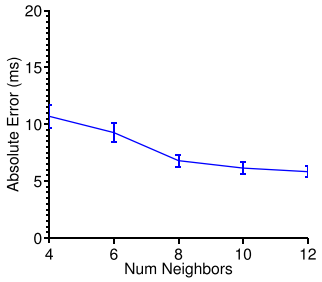


Fig. 12. Absolute error as we vary numbers of neighbors in each ring.

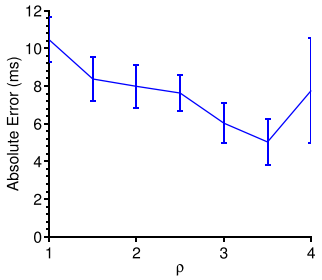


Fig. 13. Absolute error as we vary the inframetric parameter ρ.

the absolute error decreases as we add more relays per ring, since we obtain a denser coverage of the latency distribution. Further, the performance improvement becomes smoother as we increase the number of relays from six to twelve, since many relays in the same ring are from proximity regions in the latency distribution.

(iv) **Inframetric** ρ

We next study how the inframetric parameter ρ, which scales the radius of the closed ball for choosing the relays, which impacts the accuracy of MCR. Figure 13 shows that the average absolute error is reduced slightly as we increase ρ from one to three, since increasing ρ expands the intervals of candidate neighbors, which generally increases the probability of locating a relay that is closer to clients. Furthermore, the confidence intervals indicate that the relative error fluctuates due to the varying relaying locations within the immediate vicinity.

(v) **Termination Threshold** β

We further test the effect of the termination threshold β on the accuracy of the relays found. The parameter β determines when we need to stop the relay-search process: If no neighbors are able to decrease the average latency to clients by at least

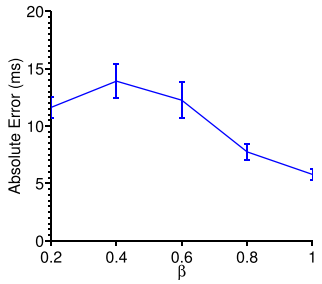


Fig. 14. Absolute error as we vary the termination threshold β.

$(1-\beta)$ times compared to that from the current node to the clients, the search process terminates.

From Figure 14, we see that increasing β gracefully decreases the absolute errors, since increasing β expands the radius of the closed ball for choosing relays, consequently, more relays are included into the candidate relays, which increases the search accuracy.

B. Experiments on the PlanetLab

Having reported the simulation results based on static RTT data sets, we next evaluate the efficiency of relay queries using the PlanetLab platform that hosts geo-distributed clients with stable connections.

1) Implementation: We have implemented a multi-thread event-driven prototype in Java. The core logic is implemented in about 3,000 lines of code. A relay exports an XML RPC interface to others to receive the relay-search request for a set of clients. All messages are sent via UDP sockets in order to avoid the connection-setup and maintenance delays.

We represent a load constraint as a declarative value. For example, the CPU load constraint can be stated as "CPU $\leq \Omega_{CPU}$", where Ω_{CPU} is the threshold of the CPU load. Measuring the load is out of the scope of this paper. For example, reading the CPU and memory loads of a server is trivial by parsing the "proc" directory on the Linux platform.

Each relay maintains a queue of incoming relay related messages. The queue gives higher priorities to the RTT measurement and relay-request messages than the gossip messages in order to minimize the queueing delays of the search process. To avoid relays being overloaded, each relay monitors its CPU load with the command-line tool "top" and piggybacks its load status to others during the gossip process.

We optimize the bandwidth cost and control the waiting time of MCR based on the hybrid-ranking policy proposed by HybridNN [18]. Briefly, each server maintains a network coordinate via the Vivaldi network coordinate method [34] and each server periodically adjusts its own coordinate with respect to a sampled neighbor's position during the gossiping process for the maintenance of the MCRing. Next, the first relay that receives a relay-search request, computes a stabilized Vivaldi network coordinate for each of these clients based on candidate neighbors' coordinates and direct probes from them to clients. These coordinates are embedded into the relay-search request message and are forwarded to the next-hop relay and henceforth. Other relays in the forwarding path then

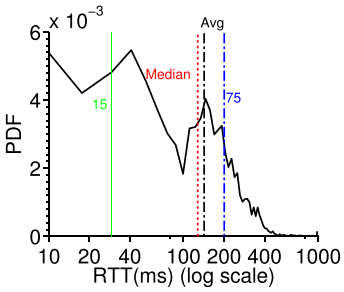


Fig. 15. The PDF of the all-pair RTT values between PlanetLab nodes.

estimate the average RTT from the candidate neighbors to clients using the Vivaldi network coordinate distances. Since Vivaldi's prediction inherently incurs some error, we select k (four by default) neighbors whose coordinate distances are top-k closest to clients, finally, we let these top-k neighbors probe RTTs towards clients to select the one with the minimum average RTT.

2) Setup: Since we are unable to access the end hosts in home networks, we use a disjoint set of PlanetLab servers as the end hosts. These servers are suitable to model the relays in geo-distributed data centers and the stable end hosts in the enterprise networks, however, they do not represent the end hosts in home networks.

We choose 173 servers from distinct sites as relays, and select another 412 servers uniformly at random as clients. The measurements last for 24 hours. We keep all relays online during the measurement. We measure the pairwise RTTs among the relays and from the relays to the clients. We compute the ground-truth relays for clients with the raw RTT samples by using the `Ping` measurements (denoted as *Direct*). Since the pairwise delays between PlanetLab machines vary dynamically, for each node pair, we use the median value of RTT samples to represent the stationary pairwise RTT. Figure 15 shows the PDF of the pairwise RTT values.

3) Results: Accuracy: We compare MCR with Meridian and iPlane [35], [36]. One of the motivating applications of iPlane was to locate the optimized relays for Skype clients [35]. For iPlane, we obtain pairwise RTTs between all-pair node from the XML RPC interface. We then compute the relay that is nearest to a set of clients. We set the same parameters for MCR and Meridian as that in the Simulation section.

The results are shown in Figure 16(a). We see that MCR has significantly lower errors than Meridian. This is because Meridian is easily trapped at local minimum due to the poor coverage in the latency space by its concentric ring, while MCR is able to locate close to optimal relays thanks to the MCRing that captures typical clusters in the latency distribution. MCR still incurs a small degree of errors, since the MCRing may not store the optimal relays towards clients. For iPlane, in around 30% of the cases, the average latency is 20 ms higher for $L = 2$ and 40 ms higher for $L = 10$, since iPlane does not use on-demand probes to find the optimized relay.

Query Time: We evaluate the time required by individual queries for MCR and Meridian. Figure 16(b) plots

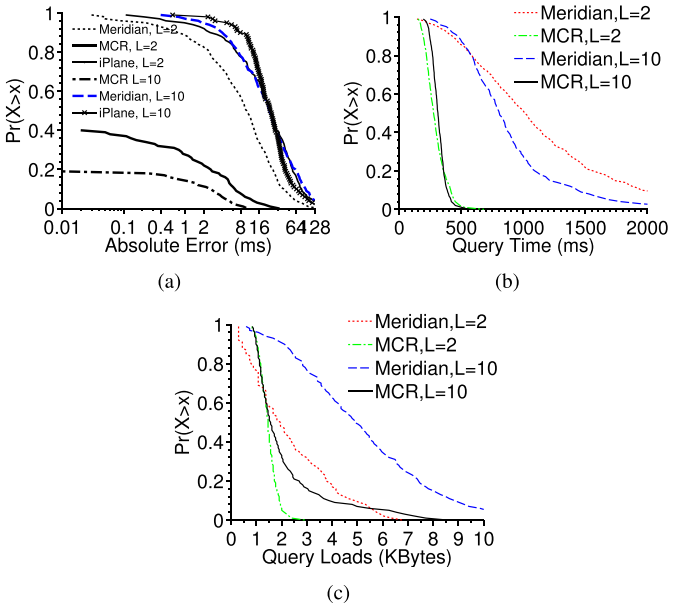


Fig. 16. The CCDFs of the absolute errors for Meridian, iPlane and MCR and the query time and the overhead for MCR and Meridian on the PlanetLab. (a) Absolute errors. (b) Query time. (c) Query overhead.

the distributions of query time of MCR and Meridian. Around 99% of MCR's queries terminate in 500 ms, while more than 95% of Meridian's requests need over 500 ms to complete. Since Meridian requires direct probes from all candidate relays to clients, some candidate relays that are far from the client take much longer time to send the response to the requesting node. As a result, the total search period is prolonged by these nodes. On the other hand, MCR avoids most direct probes by delay predictions.

Query Overhead: We define the load of a query as the total size of the transmitted packets. We plot the loads of MCR and Meridian in Figure 16(c). The load of MCR is significantly lower than that of Meridian. For $L = 2$, more than 90% of MCR queries need less than two KBytes, while in more than 50% of the cases the load of Meridian is more than two KBytes, which is due to the large size of the candidate relay set. Moreover, for $L = 10$, MCR has significantly much lower loads than Meridian. Therefore, the delay estimation of MCR substantially reduces the query overhead.

VII. Conclusion

Low-latency relay communication will be increasingly important. Existing approaches typically use concentric rings that favor nodes within the immediate vicinity of the current node. Unfortunately, as clients are unknown a priori, the optimal relay is generally outside of the immediate vicinity.

We address these limitations by proposing a structure-aware overlay called MCRing that seeks to maximize the probability of locating a relay closer to clients. We propose a distributed method to recursively locate a relay node that meets the load constraint and is closer to clients. We prove that we are able to achieve close to optimal results based on the theoretical framework of the doubling-dimension, which is more general

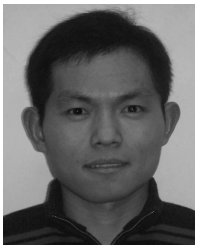
than the growth metric. Finally, experimental results and a PlanetLab deployment show that our approach has a mean error that is several times lower than those of the state-of-the-art methods.

ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers for their constructive comments.

REFERENCES

[1] S. Savage *et al.*, "Detour: Informed Internet routing and transport," *IEEE Micro*, vol. 19, no. 1, pp. 50–59, Jan./Feb. 1999.
[2] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient overlay networks," in *Proc. SOSP*, 2001, pp. 31–41.
[3] C. Lumezanu, R. Baden, D. Levin, N. Spring, and B. Bhattacharjee, "Symbiotic relationships in Internet routing overlays," in *Proc. NSDI*, 2009, pp. 467–480.
[4] S. Agarwal and J. R. Lorch, "Matchmaking for online games and other latency-sensitive P2P Systems," in *Proc. SIGCOMM*, 2009, pp. 315–326.
[5] Y. Fu, Y. Wang, and E. Biersack, "A general scalable and accurate decentralized level monitoring method for large-scale dynamic service provision in hybrid clouds," *Future Generat. Comput. Syst.*, vol. 29, no. 5, pp. 1235–1253, 2013.
[6] G. Caizzone, A. Corghi, P. Giacomazzi, and M. Nonnoi, "Analysis of the scalability of the overlay Skype system," in *Proc. ICC*, May 2008, pp. 5652–5658.
[7] R. Chen, I. E. Akkus, and P. Francis, "SplitX: High-performance private analytics," in *Proc. SIGCOMM*, 2013, pp. 315–326.
[8] J. Sherry *et al.*, "Making middleboxes someone else's problem: Network processing as a cloud service," in *Proc. SIGCOMM*, 2012, pp. 13–24.
[9] A.-J. Su, D. R. Choffnes, A. Kuzmanovic, and F. E. Bustamante, "Drafting behind Akamai," in *Proc. SIGCOMM*, 2006, pp. 435–446.
[10] D. Choffnes and F. E. Bustamante, "On the effectiveness of measurement reuse for performance-based detouring," in *Proc. INFOCOM*, Apr. 2009, pp. 693–701.
[11] D. Sontag, Y. Zhang, A. Phanishayee, D. G. Andersen, and D. Karger, "Scaling all-pairs overlay routing," in *Proc. CoNEXT*, 2009, pp. 145–156.
[12] (May 2017). *Amazon-Cloudfront*. [Online]. Available: https://aws.amazon.com/cn/cloudfront
[13] (2017). *Akamai-Sureroute*. [Online]. Available: https://developer.akamai.com/learn/Optimization/SureRoute.html
[14] B. M. Maggs and R. K. Sitaraman, "Algorithmic nuggets in content delivery," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 3, pp. 52–66, Jul. 2015.
[15] C. Lumezanu, R. Baden, N. Spring, and B. Bhattacharjee, "Triangle inequality variations in the Internet," in *Proc. IMC*, 2009, pp. 177–183.
[16] F. Dabek, R. Cox, M. F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," in *Proc. SIGCOMM*, 2004, pp. 15–26.
[17] B. Wong, A. Slivkins, and E. G. Sirer, "Meridian: A lightweight network location service without virtual coordinates," in *Proc. SIGCOMM*, 2005, pp. 85–96.
[18] Y. Fu, Y. Wang, and E. Biersack, "HybridNN: An accurate and scalable network location service based on the inframetric model," *Future Generat. Comput. Syst.*, vol. 29, no. 6, pp. 1485–1504, 2013.
[19] Y. Fu, Y. Wang, and X. Pei, "Towards latency-optimal distributed relay selection," in *Proc. IEEE/ACM CCGRID*, May 2015, pp. 433–442.
[20] P. Fraigniaud, E. Lebhar, and L. Viennot, "The inframetric model for the Internet," in *Proc. INFOCOM*, Apr. 2008, pp. 1085–1093.
[21] C. Lumezanu, R. Baden, N. Spring, and B. Bhattacharjee, "Triangle inequality and routing policy violations in the Internet," in *Proc. PAM*, 2009, pp. 45–54.
[22] A. Nakao and L. Peterson, "Scalable routing overlay networks," *ACM SIGOPS Oper. Syst. Rev.*, vol. 40, no. 1, pp. 49–61, 2006.
[23] J. Jiang *et al.*, "VIA: Improving Internet telephony call quality using predictive relay selection," in *Proc. SIGCOMM*, 2016, pp. 286–299.
[24] C. Ly, C.-H. Hsu, and M. Hefeeda, "Improving online gaming quality using detour paths," in *Proc. Multimedia*, 2010, pp. 55–64.
[25] J. Ledlie, P. Gardner, and M. I. Seltzer, "Network coordinates in the wild," in *Proc. NSDI*, 2007, pp. 299–311.
[26] V. Vishnumurthy and P. Francis, "On the difficulty of finding the nearest peer in P2P systems," in *Proc. IMC*, 2008, pp. 9–14.
[27] Y. Mao, L. K. Saul, and J. M. Smith, "IDES: An Internet distance estimation service for large networks," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 12, pp. 2273–2284, Dec. 2006.
[28] R. Xu and D. Wunsch, "Survey of clustering algorithms," *IEEE Trans. Neural Netw.*, vol. 16, no. 3, pp. 645–678, May 2005.
[29] P2PSim. (Oct. 2010). *The P2PSim Project*. [Online]. Available: http://pdos.csail.mit.edu/p2psim/kingdata/
[30] K. P. Gummadi, S. Saroiu, and S. D. Gribble, "King: Estimating latency between arbitrary Internet end hosts," in *Proc. IMW*, 2002, pp. 5–18.
[31] B. Zhang *et al.*, "Measurement-based analysis, modeling, and synthesis of the Internet delay space," *IEEE/ACM Trans. Netw.*, vol. 18, no. 1, pp. 229–242, Feb. 2010.
[32] D. R. Choffnes, M. Sanchez, and F. E. Bustamante, "Network positioning from the edge—An empirical study of the effectiveness of network positioning in P2P systems," in *Proc. INFOCOM*, Mar. 2010, pp. 291–295.
[33] Wikipedia. (Jan. 2011). *Sample Skewness*. [Online]. Available: http://en.wikipedia.org/wiki/Skewness
[34] G. Wang, B. Zhang, and T. S. Ng, "Towards network triangle inequality violation aware distributed systems," in *Proc. IMC*, 2007, pp. 175–188.
[35] H. V. Madhyastha *et al.*, "iPlane: An information plane for distributed services," in *Proc. OSDI*, 2006, pp. 367–380.
[36] H. V. Madhyastha, E. Katz-Bassett, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "*iPlane Nano*: Path prediction for peer-to-peer applications," in *Proc. NSDI*, 2009, pp. 137–152.



Yongquan Fu received the B.E. degree in computer science and technology from Shandong University, Jinan, China, in 2005, and the M.S. and Ph.D. degrees in computer science and technology from the National University of Defense Technology, Changsha, China, in 2007 and 2012, respectively. Since 2013, he has been with the Science and Technology Laboratory of Parallel and Distributed Processing, College of Computer, National University of Defense Technology, where he is currently a Lecturer. His research interests include network measurement, social networks, and distributed systems.

Ernst Biersack received the degree in computer science from the Technische Universität München and the University of North Carolina at Chapel Hill, the Dipl.Inform. (M.S.) and Dr.rer.nat. (Ph.D.) degrees in computer science from the Technische Universität München, Munich, Germany, and the Habilitation à Diriger des Recherches degree from the University of Nice, France. From 1989 to 1992, he was a member of Technical Staff with the Computer Communications Research Group of Bell Communications Research, Morristown, NJ, USA. From 1992 to 2014, he was a Professor with Eurecom, Sophia Antipolis, France.