

On The Difficulty of Finding the Nearest Peer in P2P Systems

Vivek Vishnumurthy and Paul Francis

Department of Computer Science, Cornell University, Ithaca, NY 14853

{vivi, francis}@cs.cornell.edu

Abstract

Finding the nearest peer, in terms of latency, is an important problem in many Internet applications. In this paper, we argue that solutions that only examine inter-peer latencies as part of their operation will find it infeasible, in certain commonly occurring scenarios, to discover the nearest peer in P2P systems. The difficulty arises out of the way the last hop is typically laid out in the Internet, where a single PoP (point of presence) belonging to an ISP provides connectivity to numerous client networks. This setup leads to a large number of peers being at about the same latency from one another, which presents a serious obstacle when a peer tries to discover another peer residing in the same network as itself. We use large-scale measurements over hosts in the Azureus P2P network and DNS servers to show that this condition does occur in real settings, and use simulations of the Meridian closest-server algorithm to show that the condition does indeed lead to difficulty in finding the exact-closest peer. We propose different possible approaches to address this issue, and show using a preliminary evaluation that one of these is very promising.

1 Introduction

In many peer-to-peer applications, it is beneficial for communicating peers to be close to each other. For example, in online games with direct interaction between gamers, user perceived experience closely depends on the latency between the interacting hosts. In first person shooter (FPS) games, for instance, an increase of latency from 20 to 40 milliseconds noticeably degrades user-perceived performance [1]. Many P2P games in fact only work with the high bandwidths and low latencies seen over LANs, resulting for instance in websites devoted to organizing LAN Parties (e.g., lanpartymap.com). In P2P file-sharing applications, file downloads are faster and more efficient when peers are close to one another: downloads between peers on the same campus network may be orders of magnitude faster than between even nearby peers over the general Internet.

The problem of discovering the closest peers in terms

of latency has been an active area of research in the recent past, and a number of solutions have been proposed. Example scalable approaches include: (i) Distance-based sampling, where each peer places other peers it knows into rings or balls of varying sizes, with closer peers tracked more often than those farther away [2, 3], (ii) Solutions based on network-coordinates, where each peer is given a coordinate indicating its “position” in the system, such that the latency between any two peers can be approximated by a function of their coordinates [4, 5]. (iii) Identifier-based sampling, where each peer has an identifier, and tracks other peers with identifier-prefixes matching its own [6, 7].

All of these approaches use the measured inter-peer latencies to drive their operation. In spite of the disparity of the approaches, they all share the following mechanism: A search for the closest peer to a given peer starts off from a random peer (or a set of random peers), selects among the neighbors of those peers to find closer peers, recursing until it discovers (ideally) the desired closest peer.

For this search process to work scalably and efficiently, the following condition must hold: When a peer P_1 is handling the search for the nearest peer of peer P_2 , P_1 should be able to efficiently find a closer peer to P_2 if one exists. This paper argues that, while this condition may hold as long as all peers are relatively far apart, *they do not always hold at all points of the search when peers are very close to each other*. Specifically, the search may not ultimately discover the closest peer if the closest peer happens to be on the same campus network or extended LAN, and therefore the distance to such a peer is measured in microseconds, not milliseconds.

The problem in this case arises out of the way the “last-hop” Internet is laid out. Each ISP has some number of PoPs (Points of Presence) that are used to provide Internet access to its customers. Typically, for a given host to send a packet to any other host not in the same local (campus or LAN) network, the packet must first travel to the given host’s PoP. This is often true even if the two hosts share the same PoP and are geographically near each other. Essentially, the last-hop topology resembles a *star-network*, with the PoP as the star node.

As a result, all hosts that gain access through the same

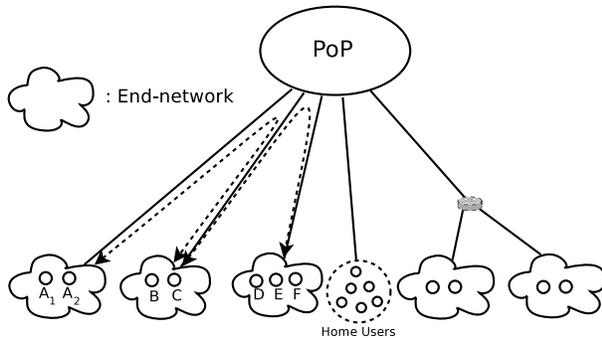


Figure 1: Typical connections from a PoP.

PoP and that are at about the same distance from the PoP end up also being about the same distance from one another. This detail makes it hard to distinguish between the different peers connected to a PoP by looking at the inter-peer latencies alone; various assumptions made by the different closest-peer algorithms, like the growth-constrained assumption, doubling assumption, and low dimensionality all fail to hold around the peers connected to a PoP. This transforms the search into a brute-force probing of the peers connected to the PoP, making it hard to scalably discover the one other peer in the same campus network from all the different peers connected to the same PoP.

This inability to find the nearest peer represents a significant “opportunity cost”: Peers that share the same extended LAN have latencies an order of magnitude smaller, and bandwidths an order of magnitude larger, than those in different networks. The ability to discover peers in the same extended LAN therefore translates to a similar order of magnitude improvement in performance of the application (e.g., gaming, P2P streaming, file-sharing), and in many cases may make the difference between being able to run a given application at all. Also, among applications like P2P streaming and file-sharing, significant savings in bandwidth costs are achieved if bulk data transmission happens between peers in the same network, rather than across the network boundary.

The focus of this paper, then, is to try to better understand this phenomenon and its implications for proximity systems. In Section 2, we provide a detailed explanation of how the large number of relatively equidistant peers served by a PoP poses a problem for closest-peer finding algorithms. We then present large-scale latency measurements over DNS-servers and real (Azureus) P2P end-hosts to indicate that this condition does indeed happen to a non-negligible extent in real scenarios (Section 3). We next use simulations of Meridian [3], a successful nearest-server finding algorithm, to demonstrate the difficulty caused by the condition in finding the nearest peer (Section 4). In Section 5, we suggest different possible approaches to tackle this issue: these approaches explicitly

or implicitly search for peers that are *topologically* close to them. We conduct a brief evaluation of one of these approaches and show that it is very likely to succeed in real settings. We describe related work in Section 6, and conclude in Section 7.

2 The Last-Hop Clustering Effect in the Internet

The Internet “last hop” provides access to end-hosts: ISPs deploy PoPs at well-populated areas, and run physical connections from end-hosts or networks of end-hosts to routers in nearby PoPs.

Figure 1 shows a typical graph of connections from the PoP. We use the term *end-network* to denote a network of end-hosts all in the same geographic location, e.g., LANs, extended LANs, and campus and corporate networks. An end-host’s *local-network* is the end-network that it resides in. It is possible that end-hosts are not part of an end-network; these would typically be hosts at homes (broadband / DSL / dial-up users). Looking at Figure 1, connections funnel in from the end-hosts and end-networks, possibly merging as they get closer to the PoP.

Suppose now that a message is sent from one of the end-hosts served by the PoP to another. We assume here that if the path from the message-source to the PoP and the path from the message-destination to the PoP share a closer upstream router than the PoP, then the message would only need to go up to the common router and then down to the destination. If both the source and the destination are in the same end-network, we assume that the message would be routed entirely within the end-network and that the corresponding latency would be much smaller than if the message had to traverse different end-networks. If the paths do not share a closer router than the PoP, and the source and destination hosts are in different end-networks, then the message needs to go all the way up to the PoP and then back to the destination. Measurements in Section 3.1 validate these assumptions.

In the following text, we restrict our attention to those end-networks where messages sent from a host in one end-network to a host in another end-network need to traverse the PoP, i.e., end-networks whose paths to the PoP share no common routers. Within this set of end-networks, we only consider those end-networks that are at about the same latency from the PoP – for our purposes, we consider two end-networks to be at about the same latency from the PoP if the latencies are close enough to each other that nearest-peer algorithms cannot reliably distinguish peers based on the difference between these latencies. We call the set of hosts in this set of end-networks the PoP’s *cluster*. Now, if the only way to distinguish between different hosts is based on the latencies to the hosts,

the above construction results in the following properties: (i) Any two hosts inside the cluster appear indistinguishable to any host outside the cluster, (ii) Similarly, any two hosts, say A_1 and B , that are inside the cluster, appear indistinguishable to any host C that is also inside the cluster, but outside the end-networks of A_1 and B (see Figure 1).

Now say there is a P2P network that consists of a few hosts from each of the end-networks in the cluster, and that each newly joining peer wants to find its closest peer. We assume that the closest-peer algorithm used here initiates a closest-peer *query* at a random peer when a new peer enters the system. In line with previously proposed solutions, we assume that the peer currently handling the query selectively probes other peers it knows in order to find a peer that is closer to the new peer. This is repeated until the closest peer is found. We assume here that the only information about a peer that the algorithm uses is its latencies to other peers (or non-peer nodes), again in line with previous solutions to this problem.

Assume now that peer A_1 has already joined the P2P system, and that peer A_2 now enters the system (see Figure 1). The closest-peer query for A_2 starts off from a random host, progressively finding peers closer to A_2 , and might eventually reach one of the peers (say C) inside the cluster. Because C is virtually a randomly picked node from the entire cluster, it is likely to be not in A_1 's local network. Ideally, the closest-peer query would eventually reach A_1 , finding it (A_1) as A_2 's closest peer. But from C 's point of view, all peers in the cluster (other than those in C 's local network) appear identical to one another: For instance, C cannot tell which of peers A_1 , D , and E , all inside the cluster, is closer to A_2 . Measurements to nodes outside the cluster are of no use here, since all peers inside the cluster appear to be at the same latency from any node outside the cluster. So the best C can do now is to hand off the query to some other peer in the cluster, in the hope that the other peer is closer to the target A_2 . The same holds true for all the peers that handle the query from this point on: The only "intelligence" each of these peers can employ in choosing the next peer is to forward the query to a peer not in its own local cluster. Thus we conclude that the query, if it does eventually reach A_1 , will have traversed through, on average, a number of peers equal to the number of end-networks in the cluster before it gets there. This translates to a lower bound on the number of latency "probes" performed as well: Since A_2 is a new peer entering the network, for a peer to tell if it is the closest peer to A_2 , it has to first measure its latency to A_2 .¹

In effect, there is a phase-transition in the performance of the algorithm once the query enters the cluster. Prior to entering the cluster, the algorithm might have made rapid

¹The one exception to this is coordinate-systems, which do not need explicit latency probes for each new latency estimate. We discuss coordinate-systems in Section 2.2.

progress in finding closer and closer peers, but once it enters the cluster, it is stuck trying to probe peers in the different end-networks in a brute-force manner. This means that when the number of end-networks in the cluster is large, finding the closest peer in the same end-network might be infeasible, since it requires a brute-force search through the different end-networks.

2.1 The Clustering Condition

The above line of reasoning is unchanged if we replace the PoP by any set of nearby routers (with negligible latencies between one another). This is important from the point of view of measurement, since accurately identifying a PoP is a hard problem. So the requirements for a cluster as described above are as follows:

1. The cluster is made of a large number of peers in different end-networks.
2. Any message sent over the Internet from a peer in one end-network of the cluster to a peer in another end-network of the cluster passes through at least one router that is part of the *cluster-hub*, a set of close-by routers.
3. All end-networks in the cluster are at about the same latency from the cluster-hub. Again, by "about the same latency", we mean that the latencies are close enough that the nearest-peer algorithm being used cannot reliably use the differences in these latencies to tell apart the different peers. How close enough they need to be depends on the particular algorithm being used.

We denote this the *clustering condition*. If a set of peers satisfies the clustering condition, it will be hard to find the closest peers to peers in the cluster. Measurements presented later in the paper, in Section 3, indicates that the clustering condition does occur in real settings with non-negligible probability.

2.2 Common Assumptions Behind Nearest-Peer Algorithms

Many of the previously proposed nearest-peer algorithms make assumptions about the inter-peer latency distribution, allowing them to be need only a provably small number of latency measurements. We now examine a few such commonly used assumptions, and illustrate how they fail to hold under the clustering condition.

Growth Constrained Metrics: The space in which the peers are located is said to be *growth-constrained* if the following condition holds: Given any peer P , and latency l , the number of all the peers within latency $2l$ from P is not significantly larger than the number of all the peers

within latency l from P [2]. In other words, if one were to draw a plot that has latency from peer P as the x-axis and the number of peers at the corresponding latency as the y-axis, the growth-constrained assumption stipulates that this curve cannot have sudden spurts. When the nearest peer to peer P is to be found, one can start from a random peer and zero in on the closest peer by repeatedly probing neighbors and progressively finding closer peers. Progress is ensured by the growth-constrained assumption, as at any point in the search, each peer is assured of having enough neighboring peers that are closer to the target peer than itself. Karger-Ruhl’s algorithm [2], and Tapestry [6] are among the nearest-peer algorithms that make the growth-constrained assumption.

Under the clustering condition, however, the space around the cluster does not conform to the growth-constrained assumption: Given a peer P inside an end-network in the cluster, we see that there is a small number of peers at very small latencies from P , and an empty space not occupied by any peers for a significant distance. This is immediately followed by a well-populated region containing other peers in the cluster. If the other peers in the cluster are between latencies l and $l + \delta$ from peer P , and $\delta \leq l$, the number of peers that are within latency $2l$ from P is significantly larger than those within latency l , thus violating the growth-constrained assumption.

Doubling Assumption: A set of peers is said to be covered by a *ball* of radius r if the latency between any two peers in the set is less than or equal to $2r$. Under the doubling assumption, any set of peers covered by a ball of radius r can be covered by a small number of balls of radius $\frac{r}{2}$. The doubling assumption is more general than the growth-constrained assumption: A space that satisfies the growth-constrained assumption also satisfies the doubling assumption [8]. The doubling assumption suggests the following approach to find the nearest peer: Say peer B wants to find the nearest peer to peer A . If both A and B are covered by a ball $ball_{AB}$, there should be a small number of smaller balls that cover the set covered by $ball_{AB}$. If B now can find some peer inside the smaller ball that covers A , progress is achieved. The Meridian nearest server algorithm [3] makes the doubling assumption.

The doubling assumption also fails under the clustering condition. Consider the smallest ball that covers all the peers in a cluster: The radius of this ball is the same as the latency of the different peers to the common upstream router(s). Any ball of half this radius would cover only those peers in a single end-network. Thus the number of smaller balls required to cover the larger ball is on the order of the large number of end-networks in the cluster, thereby violating the doubling assumption.

Low Dimensionality: Under this assumption, the latency-space can be embedded with very little error into a low-dimension space, usually Euclidean. Peers then have

coordinates assigned to them, and latencies between any two peers can be estimated using the coordinates without having to resort to active measurements between the two peers. Example approaches here include Mithos [4] and PIC [5]. However, where the clustering condition holds, the latency-space around the cluster has high dimensions: the number of dimensions is on the order of the number of end-networks in the cluster.

2.3 Behavior of Sample Nearest-Peer Algorithms Under the Clustering Condition

We now examine how a few specific nearest-peer finding algorithms would fare under the clustering condition: Meridian [3] is an algorithm designed to find the closest node from among several nodes (e.g., servers) to a given target (e.g., a client that wants to find the closest server from a set of servers). Meridian builds an overlay of the participant nodes, with each node organizing other nodes into rings of different radii: Other nodes close to a given node will occupy the nearer rings of the given node, and vice-versa. Each ring can have up to a maximum number of nodes. Members of a ring are also chosen so that they have a high hypervolume. In order to find the closest node to a given target, Meridian initiates a query starting from a random node. The node currently processing the query measures its latency to the target, and asks the nodes in its rings that it knows are at about the same latency to itself to measure their latencies to the target. The query is then forwarded to the node with the minimum distance to the target. The query terminates when the current node can find no closer node to the target than itself. When the underlying latency space satisfies the doubling assumption, the fact that the members of a ring have high hypervolume and thus are far apart from one another helps Meridian efficiently pick a closer node to the target.

To use Meridian to find the closest peer to a given *peer*, we would just have to run a Meridian query with the given peer as the target. Under the clustering conditions described above, the query would eventually reach one of the peers, say peer P in the cluster. Since almost all of the other peers (barring those in P ’s end-network) are at the same latency to P as P is to the target peer, the set of peers next asked to measure their latencies to the target is practically just a randomly chosen set from the entire cluster. The hypervolume maximization does not help here, since this space does not satisfy the doubling assumption: Any set of randomly chosen peers from the cluster has about the same hypervolume, so almost all peers in the cluster would be equally good (or bad) choices as ring members. Thus the only way the query would reach the correct end-network is by random chance. Accordingly, the query will terminate quickly, and likely not in the same end-network as the target.

The PIC [5] algorithm assigns each peer a multi-dimensional Euclidean coordinate that approximates its “position” in the latency-space. In order for a peer to find its closest peer, it first computes its (rough) coordinates, and then launches multiple greedy walks aimed at finding closer peers: At each hop of the walk, the walk chooses the closest neighbor as predicted by the respective coordinates as the next hop². However, under the clustering condition, to assign coordinates to each peer without error would need an impractically huge number of dimensions. With a small number of dimensions, all peers within a cluster would end up having almost the same coordinates, thus making it impossible to tell them apart, and ensuring the search for the nearest peer most likely does not reach the target end-network.

3 Clustering Condition in the Internet

In this section, we verify that the clustering condition occurs among real peers in the Internet, and in so doing, validate assumptions made in the previous section. To investigate the existence of the clustering condition, we use a large set of IP addresses of peers in the Azureus P2P network, taken from Ledlie et al’s study [9, 10]. We run traceroute from multiple geographically distributed vantage points to identify clusters of peers and their cluster-hubs. To directly check that messages sent from one peer in the cluster to another traverses the cluster-hub, however, we would need to have control over the peers. Since this is not the case, we instead use an alternate measurement setup using recursive DNS servers to show this property. We use the King technique [11] to measure the latency between pairs of DNS servers in a cluster, and compare this latency with the sum of latencies from the respective DNS servers to the cluster-hub. We use experiments over the DNS servers to also verify the important assumption that latencies within end-networks are significantly smaller than latencies across different end-networks.

We present our DNS server latency measurements next, in Section 3.1, and then present the clustering results over Azureus peers in Section 3.2.

3.1 Latency Measurement Results over DNS servers

We first look at the question of how well predicted latencies between close-by DNS servers match the actual latencies between them. We use a set of about 22,000 recursive

²PIC also has a variant where the new peer’s coordinates are repeatedly recomputed at each step of the greedy walks, but our argument holds equally well for the variant

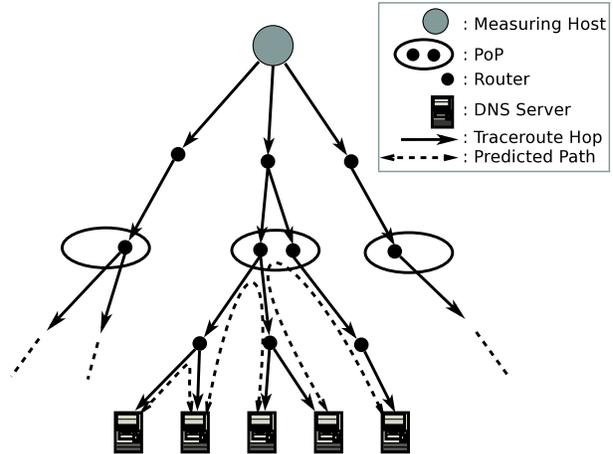


Figure 2: A sample tree of traceroutes from the measuring host. For clarity, routes are shown to be smaller than they typically are.

DNS servers, taken from Ballani et al’s study [12], as the basis for the measurements.

We use the *rockettrace* utility [13], an extension of traceroute, in the measurement here. In addition to reporting the names and IP addresses of routers on the way to the destination, *rockettrace* also annotates router names with the router’s owning AS (autonomous system) and city where the router is located. We assume that routers annotated with the same AS and city reside in the same ISP PoP. We run *rockettrace* from a single measurement host to each DNS server, and map each DNS server to its closest upstream PoP on the trace, as given by *rockettrace*. Thus, for each PoP, we are able to get the cluster of DNS servers that have the PoP as their closest upstream PoP. We then randomly pick pairs of DNS servers from each cluster, such that each DNS server appears in about 4 pairs. We measure the latency between the servers in the pair using the King technique [11]. King first measures the latency from the measurement host (the host that King is being executed on) to one of the recursive name-servers in the pair. It then sends this recursive name-server a recursive name-query for a name that the second name-server is an authoritative name server for, so the query is forwarded to the second name-server. King is thus able to estimate the latency between the two name-servers.

We predict the latency between two DNS servers in a cluster in the following manner (also see Figure 2): (i) If the *rockettrace* paths to the two servers share a closer router than the PoP to the servers (i.e., a router that is further downstream to the DNS servers than the PoP), then we predict that messages sent between the two servers to traverse up until the closest common router, and then back down to the destination. Accordingly, the predicted latency between the two routers is the sum of the laten-

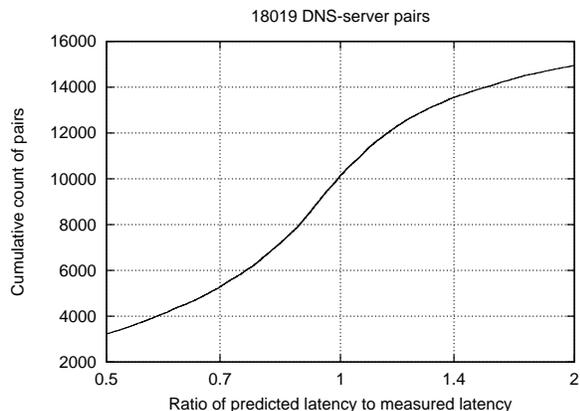


Figure 3: Cumulative distribution of the prediction measure.

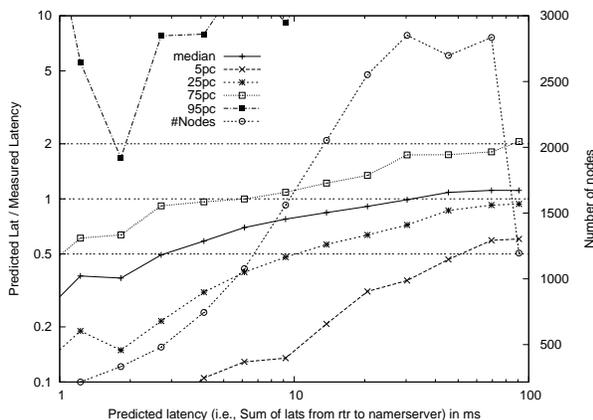


Figure 4: Tracking accuracy of prediction as a function of the predicted latency between the pairs.

cies from the DNS servers to the common router. We get these latter latencies using the ping tool, by subtracting latency to the closest common router from the latencies to the DNS servers. (ii) If the rockettrace paths share no closer router than the PoP, we predict that the latency between the two servers is the sum of the latencies from the servers to the PoP. The reasoning here is that routers in a PoP are quite close together, and should have negligible latencies between one another. We again measure latencies from the DNS servers to the PoP using the ping tool.

Figure 3 shows the cumulative distribution of the *prediction measure*, which we define as the ratio of the predicted latency to the measured latency between a pair of DNS servers. The closer this figure is to 1, the better the accuracy of prediction is. The plot does not include pairs made of DNS servers from the same domain: Such servers are highly likely to be authoritative name-servers for the same names, so the recursive queries used by King may not be forwarded to the second name-server, making King

unusable in this scenario. We also discard entries where the computed latencies between a DNS server and the relevant router or PoP turned out to be negative (as a result of the subtraction of the latency to the router from the latency to the DNS server). Finally, we exclude pairs where the DNS servers are more than 10 hops away from their closest common upstream PoP or common router, and pairs where the predicted latency between the DNS servers is more than 100 ms. This is because DNS servers that are farther away will probably have alternate shorter paths between them. After these eliminations, we have a residual set of 18019 DNS server pairs, and Figure 3 shows that about 11700 of these, i.e., about 65% of the tested pairs, have prediction measure between the range of 0.5 and 2.

Figure 4 shows the prediction measure (median, and percentile values) as a function of the predicted distance between the pairs. The plot is essentially a scatter-plot of the prediction measure versus the predicted latency, but where (for ease of understanding) we group sample points from nearby predicted latencies into a single bin with a representative predicted latency value, and where we display the median and percentiles of the prediction measure for the sample points that fall in the respective bin.

There is a definite trend visible in the plot that indicates that the prediction measure increases with the predicted latency. In other words, as the predicted latency increases, the measured latency decreases in comparison to the predicted latency. We believe this trend arises because of two reasons: At low latencies, the lag involved at the DNS servers executing the King measurements is likely to constitute a non-negligible part of the measured latency, thus leading to an artificial increase in the measured latency. On the other hand, at large latencies, it gets more likely that there are alternate paths between the DNS servers that do not traverse the common upstream router, thereby decreasing the measured latency and increasing the prediction measure. Also, DNS servers in general are more well-connected than normal end-hosts, so can be expected to have more alternate paths at large latencies.

In the argument outlined in the previous section, we had assumed that latencies between two nodes in an end-network were significantly smaller than latencies between nodes that are in the same cluster but in different end-networks. We now verify this assumption. To obtain sets of nodes that are more likely to be in the same end-network than other nodes, we assemble pairs of DNS servers sharing the same domain name. Figure 5 compares the intra-domain latency distribution among these pairs with the latency distribution among pairs of DNS servers in different domains. We obtain the two intra-domain curves in the figure by restricting the maximum number of hops between the DNS servers and the closest common upstream PoP or router to, respectively, 5 and 10. We similarly restrict the maximum number of hops in the inter-domain

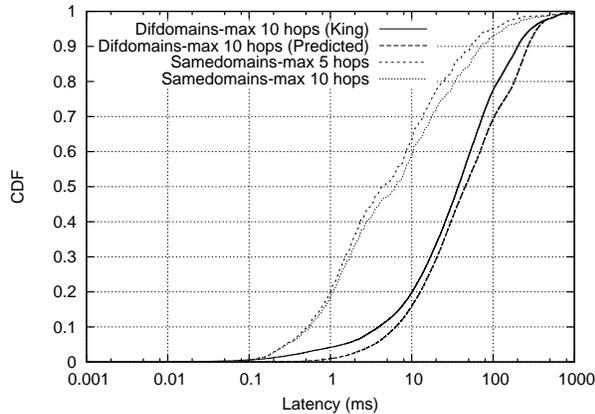


Figure 5: Comparing intra-domain latencies with inter-domain latencies.

case to 10, since we want to retain only those pairs in the same cluster. We use the predicted latencies to compute the intra-domain latency distribution, since King cannot be used here, as described earlier. We plot both the predicted and King-measured latencies for the inter-domain DNS server pairs.³

The figure shows that the intra-domain latencies are indeed much smaller (by about an order of magnitude) than the inter-domain latencies, confirming our assumption. Also, pruning the maximum number of hops from 10 to 5 results in only a modest reduction in the latencies, mainly because very few DNS servers in the intra-domain pairs are farther than 5 hops from their common upstream router. We note here that our method of compiling the intra-domain DNS server pairs is only an approximation of hosts in the same end-network; we noticed cases where the DNS servers in a pair were located in different geographic locations. We therefore expect hosts in the same end-network to have even smaller latencies than those shown in this plot.

A final aspect noticeable from the plot is that the inter-domain predicted latency distribution matches the measured latency distribution reasonably well.

Overall, the results in this section show that latencies between hosts in the same end-network are significantly smaller than that between hosts in different networks. The prediction results, while considerably accurate, are admittedly not as decisive: A non-negligible portion of the predicted distances (35%) lie outside the range of 0.5 to 2. There are factors in addition to those mentioned earlier that possibly lead to errors in the measured data: Firstly, measurements over the Internet are inherently prone to noise, so cannot be expected to give consistently accurate results. Also, rockettrace’s method of annotating routers

³There are about 500 DNS server pairs in the intra-domain distribution, and about 26000 pairs in the inter-domain latency distribution.

Vantage Point	Location
planetlab02.cs.washington.edu	Washington, USA
planetlab3.ucsd.edu	California, USA
planetlab5.cs.cornell.edu	New York, USA
planetlab2.acis.ufl.edu	Florida, USA
neu1.6planetlab.edu.cn	Shenyang, China
planetlab2.iii.u-tokyo.ac.jp	Tokyo, Japan
planetlab2.xeno.cl.cam.ac.uk	Cambridge, England

Table 1: The set of Planetlab [14] nodes used as vantage points

with information about the router’s AS and geographical information is based on the name of the router; if the name is mis-configured, this leads to erroneous results. In view of these mitigating factors, we believe that while the results are noisy, they do indicate that most of the nodes in a cluster do need to traverse the closest common router in order to communicate with one another.

We extend this finding to peers that are end-hosts (and not servers) as well: We actually would expect better prediction accuracies with peers than with DNS servers, owing to the fact that servers are likely to have more alternate connections between them.

3.2 Measurement over Azureus Client IP Addresses

We now examine the occurrence of the clustering property in the Azureus P2P network, using a set of 156,658 Azureus IP addresses collected by Ledlie et al [9, 10]. The basic method is as follows: We track each peer’s closest upstream router using traceroutes from multiple vantage points spread across the globe, produce clusters of peers that all have the same upstream router, identify the common upstream router as the cluster-hubs, measure latencies between the cluster-hub and the peers within each cluster, and further prune down the clusters to ensure all cluster peers have similar latencies to the cluster-hub.

The closest upstream router of a peer, as seen from a particular vantage point, is the last router seen on the trace from the vantage point to the peer.⁴ We retain only those peers that have the same upstream router as seen from all the vantage points.

We group peers with the same upstream router into clusters. Table 1 shows the set of vantage points used. The fact that these are well-distributed across the globe, and the DNS-server measurement results from Section 3.1 indicate that the common upstream router is on the route between any two peers in the cluster. We thus choose the common upstream router as the cluster-hub.

⁴We consider only valid routers here. E.g., if none of the entries in the penultimate hop of a traceroute are valid, we go up to the next hop(s) to get the closest upstream router.

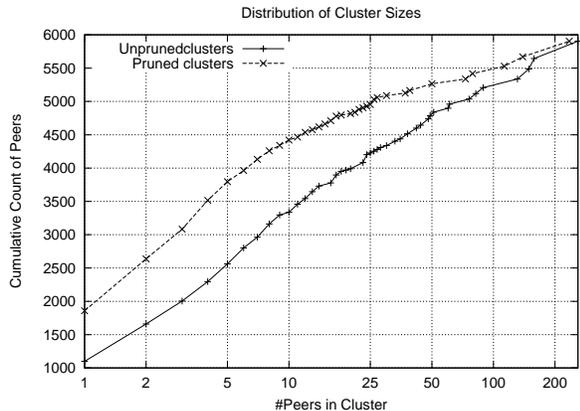


Figure 6: Distribution of cluster sizes, both before and after pruning, with 5904 peers in all.

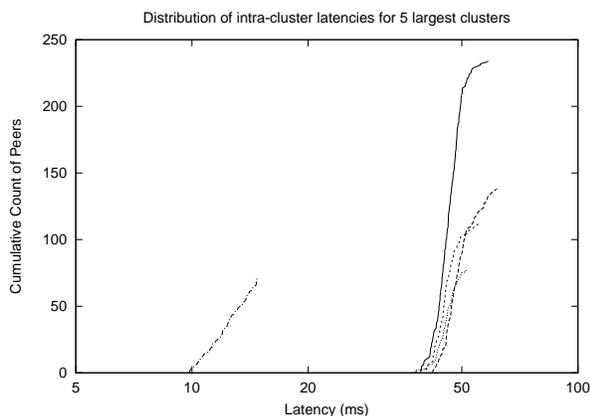


Figure 7: Distribution of latencies within clusters for the largest 5 clusters. The cluster-sizes are, respectively, 235, 139, 113, 79, and 73

To measure the distribution of latencies from the cluster-hub of the clusters of peers to each of the peers in the cluster, we should be able to first measure the latencies to the peers themselves. But ping and traceroute, the usual tools of choice, mostly fail here: Most peers do not respond to either ping or traceroute with valid latencies. Since the peers here are Azureus clients that communicate over TCP and use a well-known port (6881), we instead measure the latency to a peer as the time it takes to complete a TCP ‘connect’ to the port at the peer; we call this the “TCP-ping”. Out of the 156,658 total IP addresses in the original list, only 5904 remained that responded to the TCP pings or traceroutes and had a unique upstream router as seen from all the vantage points. We group these peers into clusters and find the latency distribution within each cluster: We launch TCP pings from the same vantage points as seen above to get latencies to the peers. And we use the appropriate entry from the traceroute output as

the latency to the cluster-hub, and subtract this from the latencies to the peers to compute the latencies from the cluster-hub to the peers in each cluster.

With the above formation of clusters, it is possible that the hub-to-peer latencies might vary widely within the cluster. So we further pare down the clusters, ensuring that within each cluster, the hub-to-peer latencies are all within a factor of 1.5 from one another. The exact extent of similarity in hub-to-peer latencies that leads the cluster-peers to be indistinguishable in the eyes of a nearest-peer algorithm of course depends on the particular algorithm itself; we use the factor of 1.5 here as an approximation of this.

Figure 6 shows the cumulative distribution of cluster sizes, both before and after the pruning step described above. About 16% of the peers are in (pruned) clusters of size 25 or larger. As a sample of the inter-peer latency distribution within clusters, Figure 7 shows the distribution of latencies from the cluster-hub to the peers in the cluster for the largest 5 pruned clusters. The latency distribution shown here indicates that most peers in the displayed clusters are in different end-networks. These results show that even the small sample of 5904 peers has a non-negligible fraction of the population in clusters that satisfy the clustering condition: they have peers spanning reasonably large numbers of end-networks, and have all peers at similar latencies from one another. Thus new peers sharing end-networks with peers in the cluster will find it hard to discover their closest peers.

We note here that the measurements presented in this section are not (and are not intended to be) an accurate quantitative evaluation of the exact extent of occurrence of the clustering condition – indeed it is almost impossible to do this, without explicit co-operation from the participant peers. Instead, these results should be taken as an indicator that the clustering condition does exist to a non-negligible degree, and that designers of latency-sensitive applications need to keep this in mind.

4 Meridian Simulations under the Clustering Condition

Earlier, in Section 2, we argued analytically that the different nearest-peer algorithms would find it difficult under the clustering condition to find exact-closest peers. We now use simulations of the Meridian algorithm to help verify this argument. We use the Meridian simulator used in the Meridian paper [3] for the simulations.

To simulate the clustering condition in the inter-peer latency matrix, we create clusters of end-networks that in turn contain peers. Each end-network in a cluster is at a given latency from the cluster-hub of the cluster: the closer these latencies are to one another, the more the

cluster conforms to the clustering condition. Within each cluster, we set the mean latency between the cluster-hub and the end-networks in the cluster to be uniformly distributed between 4 ms and 6 ms. We use a parameter δ that quantifies the variation of latencies within a cluster – the latency of each end-network to its cluster-hub is uniformly distributed between $(1 - \delta)$ and $(1 + \delta)$ times the mean latency between the cluster-hub and the end-networks in the cluster. We use the Meridian DNS-server latency dataset [3, 15] to simulate latencies between the cluster-hubs: each cluster-hub is represented by a randomly picked DNS server from the dataset. DNS-server pairs in the Meridian dataset have a median latency of around 65 ms.

All end-networks in our simulation contain two peers each. Peers that are both in the same end-network have a latency of $100 \mu s$ between them, and identical latencies to all other peers. Two peers in different end-networks have an inter-peer latency equal to the latency between the end-networks that contain them, computed according to the latency assignment in the previous paragraph (where the path starts from one peer, goes up to its cluster-hub, across to the cluster-hub of the second peer, and down to the second peer).

The above assignment satisfies the expected gradation of latencies: latencies within an end-network are more than a magnitude smaller than latencies across end-networks, and latencies within a cluster are smaller than latencies across clusters. We are interested here in identifying recognizable trends that Meridian exhibits with changing clustering properties.

The above setup is used to build inter-peer latency matrices with about 2500 peers, out of which about 2400 randomly picked peers are picked to build a Meridian overlay. The 100 remaining peers are used as *target nodes*, where Meridian tries to find the closest peer in the overlay to chosen target nodes. In each simulation, 5000 Meridian closest-neighbor queries are launched to find the closest peer to randomly chosen target nodes. Note that the target nodes themselves do not join the Meridian overlay, thereby letting reuse of the same target multiple times. Also, since the target nodes are picked randomly from the original set of peers, it is very likely that the target shares the same end-network as some other peer in the overlay, and this peer would be the closest peer in the overlay to the target.

We ran all of the Meridian simulations with the Meridian parameter β set to 0.5, and the number of neighbors per ring set to 16, as in the Meridian paper.⁵ All the numbers presented in this section are the results of three separate simulations, each using a different inter-peer latency

⁵The β parameter in Meridian controls the trade-off between the number of messages sent as part of a Meridian query resolution and the accuracy of the result of the query.

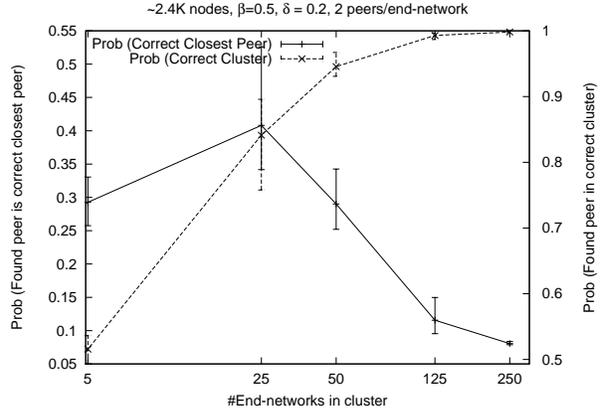


Figure 8: Meridian success rates in finding (i) the absolute closest peer, and (ii) some peer in the same cluster as the target node.

dataset.

We first look at the changing performance of Meridian with the change in the number of peers in the cluster. Figure 8 shows, as a function of the average number of end-networks in a cluster, the proportion of times Meridian is able to find the correct closest peer and the proportion of times it is able to find a peer in the correct cluster as the closest peer.⁶ The correct cluster here is the cluster that contains the target node. We set δ to 0.2 in these simulations. The accuracy of Meridian’s choice of closest peer initially improves with an increase in cluster-size, but falls off at larger sizes, while the probability of finding some peer in the correct cluster uniformly improves with cluster size. The reason for the latter behavior is that with larger cluster sizes, there are more peers from the correct cluster, improving their chance of being discovered by the Meridian queries. At the lower end of the spectrum of cluster-sizes, the accuracy of Meridian’s choice of the closest peer also improves with an increase in cluster-size, owing to an increased probability that the query enters the correct cluster in the first place. But beyond a certain point (at $x=25$ in the plot), the increased likelihood of finding the correct cluster is more than outweighed by the phase transition caused by the emergence of the clustering condition: There is less and less chance that random probing among peers inside the cluster leads the query to the correct end-network. This result shows that the probability of finding the correct closest peer indeed deteriorates when the clustering condition occurs.

We next examine the effect of variations in intra-cluster latencies on the accuracy of Meridian. The parameter δ described above captures this variation. We run Meridian simulations over a range of different values of δ , start-

⁶The plotted values are the median, minimum and maximum values across the three simulation runs.

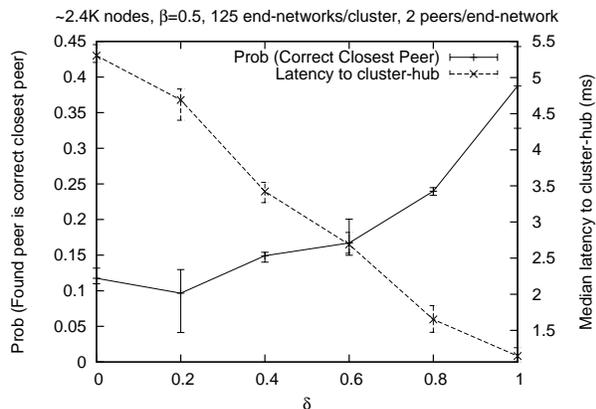


Figure 9: Meridian’s accuracy in finding the absolute-closest peer and the latency of the discovered peer from the cluster-hub as functions of δ , the variation in latencies within a cluster.

ing from $\delta = 0$, with no variation in intra-cluster latencies, to $\delta = 1$, where latencies from a peer to its cluster-hub could range anywhere between 0 and twice the average hub-to-peer latency for the cluster. Note here that the larger δ is, the less the network conforms to the clustering condition. We run the simulations with an average 125 end-networks in each cluster. Figure 9 shows the results. With an increase in δ , there is a significant improvement in Meridian’s accuracy in finding the closest peer. This is a direct result of the clustering condition holding for smaller values of δ , and its weakening at larger values of δ . For larger values of δ , the cluster could effectively be split into smaller clusters, where within each cluster, there is a much smaller variation in the intra-cluster latencies. With smaller clusters, there is a greater likelihood of random probing succeeding, thus leading to better accuracy in finding the nearest peer.

Figure 9 also shows the average latency from the cluster-hub to the peer found by Meridian, not counting those cases where Meridian actually found the correct closest peer. The latency decreases with an increase in δ . This is because for higher values of δ , there would be peers that are closer to the cluster-hub (by construction). Peers that are closer to the cluster-hub are also closer to all other peers in the cluster, so Meridian, by design, preferentially picks such peers over others. A side-effect of this is that peers closer to the cluster-hub end up being selected more often than others, increasing the load placed on them. This raises an interesting but hard-to-answer question: Given that it is hard at times to find the closest peer in the same end-network, should we aim to find the closest peer that *can* be found, keeping in mind that doing so would end up overloading a few peers? An alternative formulation would be one that encourages the discovery

of another peer in the same end-network, but relaxes the constraints if such a peer cannot be found.

Backing up however, we note that the Meridian simulation results verify our earlier argument: It is hard to find the closest peer in clusters where the clusters have a large number of end-networks and the end-networks are all at about the same latencies to the respective cluster-hubs.

5 Mechanisms to Handle Clustering Effect

The previous sections argued how it would be hard to find the exact-closest peer in large P2P systems by examining inter-peer latencies alone. We next outline three basic approaches that try to solve the problem by incorporating additional information while finding the nearest peer. At the end of the section, we give a preliminary evaluation of the easiest to deploy of these approaches.

The first approach consists of a simple expanding search within each end-network using IP multicast; this search is aimed at finding other peers in the end-network. This technique has been suggested in previous work (e.g., to help find the nearest server [16], and to find existing peers in a P2P system to help bootstrap a new peer [17]). This approach however assumes that IP multicast is enabled within each end-network and that messages multicast from one host inside the end-network is capable of reaching any other host in the end-network; the latter assumption may often be invalid in large end-networks that are themselves composed of multiple LANs or VLANs.

The second approach uses a central server inside each end-network that tracks all peers inside the end-network that are currently in the P2P system. This server could conceivably be used to track membership in multiple P2P systems. The concern with this approach, aside from the obvious one regarding its centralized nature, is that it needs a sufficiently large number of peers within each end-network to justify the setup of the membership tracking server.

The third approach needs no explicit support from the network and can be implemented in a completely decentralized fashion. This approach uses hints to the actual *location* of a newly entering peer to help find its closest peer. The two hints we consider here are (i) The new peer’s IP address, and (ii) The peer’s *Upstream Connectivity List (UCL)*, i.e., the list of routers that are at a fixed number of hops (say 5) or closer from the peer, where peers would determine their UCLs by running traceroutes to a few different locations in the Internet. The intuition here is that two peers that have matching IP address prefixes or similar UCLs are likely to be close to each other.

We note that IP address prefixes and upstream routers have both been suggested as hints to proximity in pre-

vious work. CoralCDN [18] uses upstream routers to map clients to nearby servers, and to find latency-sensitive paths in an overlay. Freedman et al [19] note that IP addresses that share the same prefix are more likely to be in the same geographic location, and OASIS [20] uses IP address prefixes to again map clients to nearby servers. In this paper however, we propose the use of UCLs and IP prefixes specifically to find the nearest peer, especially where the nearest peers share the same extended LAN.

The third approach requires a key-value mapping infrastructure to help peers find other peers with similar IP addresses or UCLs. In the UCL-based heuristic, a mapping is created for each upstream router and peers that have the router in their UCLs: the key here is the IP address of the upstream router, and the value the IP addresses of the peers that have the router in their UCLs. When a new peer enters the system, it obtains its UCL, and uses the key-value map to retrieve IP addresses of all peers that it shares upstream routers with. The new peer can now actively probe the retrieved addresses to find the closest among them. The new peer inserts its own mapping once it joins the system. This approach ensures that peers that share a close upstream router would be able to find one another, provided that the IP address of the router is visible to the peers.

The IP-prefix based approach is similar to the above, except for the fact that the key used to store the mapping is a fixed-length prefix (e.g., the /24 prefix) of the peer’s IP address.

The participant peers can themselves host the key-value maps required above, using one of several distributed hash table (DHT) designs available (Chord [21], CAN [22], Pastry [17], etc.). Many DHTs assume that keys are uniformly distributed, which may not be the case with IP addresses. In such scenarios, the IP addresses can be hashed to compute the keys to use in the system.

The hints used in the third approach, namely the UCL and the IP prefix, also have an additional (related) application beyond finding the nearest peer: They may be used in proximity-address based systems like Vivaldi and PIC [23, 5]. In these cases, the UCL (or the IP prefix) is added as an extension of the otherwise latency-based proximity address. When comparing two such composite addresses, if the UCL indicates that the nodes share an upstream router, then the nodes are considered to be close together and the proximity address may be ignored. If the two nodes do not share an upstream router, then the UCL is ignored.

Note that the three approaches listed above would be used in conjunction with existing near-peer finding algorithms (and with one another) to obtain maximum accuracy in finding the nearest peer. The third approach (based on UCL or IP-prefix) however has the advantage of being able to be deployed in a decentralized fashion, and without need for extra network support.

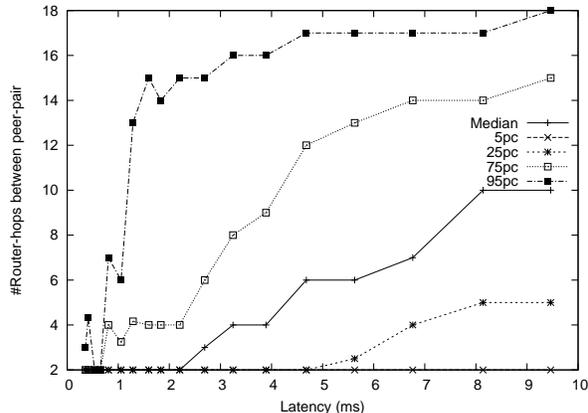


Figure 10: Inter-peer router hop-length as a function of inter-peer latency, for the UCL-based approach. The number of routers to be tracked in order to discover peers that are at a given latency range is equal to half the corresponding hop-length value.

We now give preliminary evaluations of both the UCL and IP-prefix based heuristics, using the Azureus peer-set from earlier (Section 3.2, [9, 10]). Our aim here is to investigate whether the heuristics are successful in finding nearby peers, and what the associated overheads are. We assume a perfect key-value map here for both approaches.

From the original peer-set, we retain the 22,796 peers that responded with a valid latency to either a TCP ping or a traceroute. We track the latencies along traceroutes from the Planetlab vantage points to the different peers to get an approximate adjacency matrix: the matrix includes the Azureus peers and the routers along the traceroutes that responded with valid latencies, and tracks the latencies between the different routers and those between the routers and the Azureus peers. We run the Dijkstra algorithm over this adjacency matrix to obtain a set of closest peers for each peer, and show results for peer-pairs that are closer than 10 ms to each other.

We present results for the UCL approach in Figure 10: it plots the router hop-lengths between close peer-pairs against the latencies between them. This plot is a “binned” scatter-plot of inter-peer hop-lengths versus inter-peer latencies, where sample points from nearby latencies are grouped into a single bin (similar to Figure 4). Note here that if all peers tracked upstream routers n hops away from them, they would be able to discover all peers $2n$ hops away, via the key-value map. So the fact that the bin at 3.9 ms has a median hop-length of 4 means that, in the median case, peers that make up the pairs in this bin would be able to discover each other (i.e., the other peers in the pairs) if each peer tracks its 2 closest upstream routers.

The figure shows that the UCL-based approach is indeed promising. The inter-peer hop-length grows with

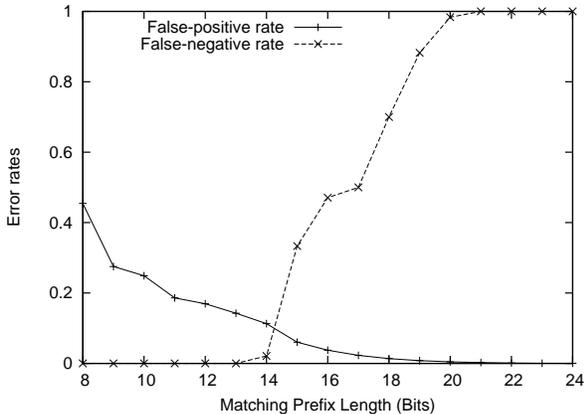


Figure 11: False-positive and false-negative rates with the IP-prefix based approach.

inter-peer latencies, implying that if the goal is to discover only very close peers, it can be achieved by having peers track only a modest number of routers: To discover peers closer than 5 ms, peers need to track 3 upstream routers each for a 50% success rate (the median case) and about 6 routers each for a 75% success rate. So this approach can be expected to perform very well when the closest peer is indeed very close-by in the general case. This also includes cases where the closest peer is in the same end-network.

On the other hand, the growing hop-length with latency has negative implications for this approach if the closest peer happens to be significantly farther away, and the goal is to still discover that closest peer. In such scenarios, we suggest coupling the above approach with traditional nearest-peer algorithms.

Figure 11 shows results for the IP-prefix based heuristic. It shows the median false-positive and false-negative rates incurred by the approach as a function of different prefix-lengths. For each peer, we compute the false-positive rate as the ratio of the number of peers that share the same IP prefix as the given peer, but are more than 10 ms away from the peer, to the total number of peers that are more than 10 ms away from the peer. Similarly, the false-negative rate is the ratio of the number of peers with a different IP prefix, but are closer than 10 ms to the peer, to the total number of peers that are closer than 10 ms to the peer. We again estimate the latency between peers as the latency along the shortest path in the traceroute-generated graph. The population-size here, i.e., the number of peers that are within 10 ms to at least one other peer, is about 2400. It is desirable of course that both the false-positive and false-negative rates are low: If the false-positive rate is high, a lot of effort is expended in further probing the nodes returned by the heuristic to actually find the few nodes that are close-by. Similarly, if the false-negative rate is high, a

large proportion of the peers that are actually close-by are never found.

Figure 11 shows, as expected, that the false-positive rate falls with more fine-grained (longer) prefixes, whereas the false-negative rate increases with longer prefixes. Unfortunately, there is no clear “sweet-spot” here: With a prefix-length of 14 bits or shorter, the false-positive rate is greater than 0.1, so at least about 250 peers need to be further probed to identify those peers that are actually close. And with larger prefix-lengths, more and more close-by peers are ignored.

The UCL-based approach, on the other hand, is not vulnerable to the above false-positive problem: In the mapping of upstream routers to end-host IP addresses, we could also embed information about the latency between the routers and the end-hosts. Two peers that share upstream routers can now form a rough estimate of their latency to each other as the sum of their latencies to the closest common router. Thus peers can discard, without further probing, other peers that are estimated to be too far away.

Since we do not control the end-hosts used in the measurements here, we are unable to empirically observe occurrences of false-negatives with the UCL approach. In practice, this will depend on the completeness of the UCL map that peers can generate: the more complete the maps are, the less is the possibility of false-negatives.

6 Related Work

The scheme by Karger-Ruhl [2] and Meridian [3] are what could be called *Distance-based sampling schemes*. Each peer here picks neighbors based on its distances to them: the concentration of neighbors is high at small latencies, and drops off at larger latencies. When peer P is handling a request to find the nearest peer to another peer N , P would forward the request to a set of neighbors at distances that are a function of the distance between P and N . Intuitively, if the request is forwarded to those neighbors that are at about the same distance as P is to N , then by random chance, one of them is likely to be closer to N than P is.

In Tapestry [6], peers arrange neighbors in different “levels”, with exponentially fewer choices for neighbors as the level increases. The idea is to pick, in each level, a fixed number of neighbors that are closest to the peer from among the available choices for the level. The levels are built up iteratively, starting from the highest level, i.e., the level with the fewest eligible neighbors that can occupy the level. The level i neighbors of peer P are chosen from among appropriate neighbors of its level $i+1$ neighbors. The peers are assumed to be in a growth-constrained metric space [2], resulting in the above iterative construction finding the closest eligible neighbors at each level. The

closest neighbor overall is the closest neighbor in the lowest level. Constrained gossiping [7] is similar, but adds periodic gossip for maintenance of neighbor information.

Practical Internet Coordinates [5] and Mithos [4] are two schemes that use *network coordinates* to estimate latencies between any two arbitrary peers, and leverage the coordinates to discover the closest peer. There have been many other network coordinate schemes that have been proposed, e.g., GNP [24], Vivaldi [23], and PCoord [25], that can potentially be used to find the closest peer as well. Distributed binning [26] is similar in vein, but instead of coordinates, uses *bin numbers* that indicate peers' relative latencies to a given set of landmarks. An assumption with these schemes is that the population of peers is embeddable into a space with a small enough number of dimensions that the coordinate scheme is accurate and practical, and the coordinates can be reliably used to find the closest peer.

Tiers [27] is a hierarchical scheme to find the nearest peer. The Tiers hierarchy consists of multiple levels: The lowest level has all the peers in the system, with nearby peers grouped into clusters. A single peer from each cluster is chosen as the cluster's representative. Each cluster representative is part of the next (higher) level, where again the member hosts are grouped into clusters and representatives chosen for these clusters. This continues until the topmost level, which has just a single cluster. When a new peer joins the system, its search for the nearest peer starts from the topmost cluster. The new peer measures its latencies to each of the nodes in the cluster, and picks the one that is closest to, and the search continues with the cluster (in the next lower level) represented by the picked peer. The search eventually reaches a cluster in the lowest level, and the nearest peer in the cluster is chosen as the nearest peer overall.

Each of the schemes outlined above fail in finding the exact closest peer (i.e., a peer in the same end-network) under the clustering condition. Recall that in the clustering condition, there are a large number of peers at about the same latency from one another. In Section 2, we explained why Meridian would be unlikely to find the nearest peer in such conditions. An almost identical argument can be made in the case of Karger-Ruhl's algorithm, for the same conclusions. Under the clustering condition, the doubling assumption [28] and growth-constriction assumption [2], required by Meridian and Karger-Ruhl's scheme respectively, are violated. The coordinate-based schemes fail here because a large number of dimensions is required to embed all the peers in the cluster, whereas the schemes assume a small number of dimensions.

In the case of Tapestry under the clustering condition, a new peer's search for its nearest peer might reach one of the peers inside its cluster, i.e., one of the levels might include neighbors in its cluster. But it is unlikely that it

will then proceed to find another peer in the same end-network as the new peer. This is because all the peers in the cluster look identical to one another, so the only way the new peer would select the correct peer is by first picking as its neighbor a peer that has the desired peer as a neighbor in the appropriate level, and the likelihood of this latter event happening is small.

Before we discuss the behavior of Tiers under the clustering condition, we need to distinguish a cluster formed by Tiers from the cluster of peers that forms the basis of the clustering condition. We refer to the former as a *Tiers-cluster*, and to the latter as a *peer-cluster*. Tiers forms multiple Tiers-clusters at the lowest level from each peer-cluster. This means that multiple peers from the same cluster also occupy higher levels in Tiers. When a new peer traverses down the hierarchy looking for its nearest peer, it will eventually select its nearest peer in the same end-network only if it picks the right cluster-representative at each step of the hierarchy. Since this essentially reduces to random choices at each step, it is unlikely to succeed in finding the exact-closest peer in the same end-network.

In contrast to the above approaches, centralized approaches using *beacon-servers* are suggested by Guyton et al [16] to find the nearest replicated server to a client and by Beacons [29] to find the nearest peer to a peer. In the former, each of the beacon servers measure their latencies to each of the servers, and the client that wishes to find its nearest server. They estimate the latency between the client and each of the servers using Hotz's metric [30] based on triangulation bounds. The server with the least estimated latency is returned as the closest server. When this approach is used to find the closest *peer*, the beacon-servers now track latencies to all peers. But under the clustering condition, this leads to most peers in the same cluster but different end-networks having almost identical latencies to all the beacon servers, since most end-networks would not have a beacon server deployed in them. It follows that all such peers are impossible to tell apart.

In Beacons, each beacon server tracks and remembers its latency to each peer. When a new peer P wants to find its closest peer, each beacon returns the set of other peers that are at about the latency to itself as P is. P then probes the peers in the returned sets, and picks the closest among these. Again, under the clustering condition, this leads to the beacon servers having the same latencies to most peers in a cluster, making them indistinguishable from one another.

7 Conclusions and Future Work

In this paper, we identified the clustering condition, and showed that it makes it expensive for latency-only based proximity methods to find extreme-nearby (same campus network) peers in the Internet. We performed large-scale

measurements over the Internet to show, with reasonable confidence, that the clustering condition does occur in real settings, and used analytical arguments and simulations to show that nearest-peer finding algorithms suffer under the condition. We listed different approaches to overcome this issue, and showed that one of them was quite promising. Overall, this paper showed that developers of latency-sensitive P2P applications need to be mindful of this factor when deploying their systems, and should employ additional mechanisms like those suggested in this paper when finding extreme-nearby peers is important.

An interesting line of future work is to determine the exact extent of occurrence of the clustering condition in particular deployed P2P systems. Doing so would however require explicit cooperation from the individual peers. Another supplementary piece of future work is to more extensively evaluate all the different mechanisms proposed in the paper to handle the clustering condition.

References

- [1] Peter Quax, Patrick Monsieus, Wim Lamotte, Danny De Vleeschauwer, and Natalie Degrande. Objective and Subjective Evaluation of the Influence of Small Amounts of Delay and Jitter on a Recent First Person Shooter Game. In *Proc. ACM SIGCOMM workshop on Network and system support for games*, 2004.
- [2] D. Karger and M. Ruhl. Finding Nearest Neighbors in Growth-restricted Metrics. In *Proc. ACM STOC*, 2002.
- [3] Bernard Wong, Aleksandrs Slivkins, and Emin Gün Sirer. Meridian: A Lightweight Network Location Service without Virtual Coordinates. In *Proc. ACM SIGCOMM*, 2005.
- [4] Marcel Waldvogel and Roberto Rinaldi. Efficient Topology-Aware Overlay Network. In *Computer Communication Review 33(1): 101-106*, 2003.
- [5] Manuel Costa, Miguel Castro, Antony I. T. Rowstron, and Peter B. Key. PIC: Practical Internet Coordinates for Distance Estimation. In *Proc. ICDCS*, 2004.
- [6] Kirsten Hildrum, John D. Kubiatowicz, Satish Rao, and Ben Y. Zhao. Distributed Object Location in a Dynamic Network. In *Proc. ACM SPAA*, 2002.
- [7] Miguel Castro, Peter Druschel, Y. Charlie Hu, and Antony Rowstron. Proximity neighbor selection in tree-based structured peer-to-peer overlays. In *Technical report MSR-TR-2003-52*, 2003.
- [8] Aleksandrs Slivkins. Embedding, Distance Estimation and Object Location in Networks. In *Ph.D. Thesis, Cornell University*, 2006.
- [9] Jonathan Ledlie, Paul Gardner, and Margo Seltzer. Network Coordinates in the Wild. In *Proc. NSDI*, 2007.
- [10] Network Coordinate Research at Harvard. <http://www.eecs.harvard.edu/~syrah/nc/>, Accessed June 2008.
- [11] Krishna P. Gummadi, Stefan Saroiu, and Steven D. Gribble. King: Estimating Latency between Arbitrary Internet End Hosts. In *Proc. ACM SIGCOMM*, 2002.
- [12] Hitesh Ballani, Paul Francis, and Sylvia Ratnasamy. A Measurement-based Deployment Proposal for IP Anycast. In *Proc. IMC*, 2006.
- [13] Neil Spring, David Wetherall, , and Tom Anderson. Scriptroute: A Public Internet Measurement Facility. In *Proc. USITS*, 2003.
- [14] Planetlab. <http://www.planet-lab.org>.
- [15] Meridian: Lightweight Positioning. <http://www.cs.cornell.edu/People/egs/meridian/>, Accessed June 2008.
- [16] James D. Guyton and Michael F. Schwartz. Locating Nearby Copies of Replicated Internet Servers. In *Proc. ACM SIGCOMM*, 1995.
- [17] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. IFIP/ACM Middleware*, 2001.
- [18] Michael J. Freedman, Eric Freudenthal, and David Mazires. Democratizing Content Publication with Coral. In *Proc. NSDI*, 2004.
- [19] Michael Freedman, Mythili Vutukuru, Nick Feamster, and Hari Balakrishnan. Geographic Locality of IP Prefixes. In *Proc. IMC*, 2005.
- [20] Michael J. Freedman, Karthik Lakshminarayanan, and David Mazieres. OASIS: Anycast for any service. In *Proc. NSDI*, 2006.
- [21] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proc. ACM SIGCOMM*, 2001.
- [22] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content-Addressable Network. In *Proc. ACM SIGCOMM 2001*, 2001.
- [23] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: A Decentralized Network Coordinate System. In *Proc. ACM SIGCOMM*, 2004.
- [24] T. S. Eugene Ng and Hui Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *Proc. IEEE Infocom*, 2002.
- [25] Li wei Lehman and Steven Lerman. PCoord: Network Position Estimation Using Peer-to-Peer Measurements. In *Proc. NCA*, 2004.
- [26] Sylvia Ratnasamy, Mark Handley, Richard M. Karp, and Scott Shenker. Topologically-Aware Overlay Construction and Server Selection. In *Proc. IEEE Infocom*, 2002.
- [27] Suman Banerjee, Christopher Kommareddy, and Bobby Bhattacharjee. Scalable Peer Finding on the Internet. In *Proc. Global Internet Symposium, Globecom*, 2002.
- [28] Anupam Gupta, Robert Krauthgamer, and James R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *FOCS*, 2003.
- [29] Christopher Kommareddy, Narendar Shankar, and Bobby Bhattacharjee. Finding Close Friends on the Internet. In *Proc. IEEE ICNP*, 2001.
- [30] S.M. Hotz. Routing information organization to support scalable interdomain routing with heterogeneous path requirements. In *Ph.D. Thesis, University of Southern California*, 1994.