Finding close friends on the Internet

Christopher Kommareddy Narendar Shankar Bobby Bhattacharjee
Department of Computer Science
University of Maryland, College Park, Maryland
{*kcr, narendar, bobby*}*@cs.umd.edu.*

Abstract

We consider the problem of finding nearby application-peers (*close friends*) over the Internet. We focus on unicast-only solutions and introduce a new scheme —*Beaconing*— for finding peers that are near. Our scheme uses distance measurement points (called beacons) and can be implemented entirely in the application-layer without investing in large infrastructure changes.

We present an extensive evaluation of Beaconing and compare it to existing schemes including Expanding Ring searches and Triangulation. Our experiments show that 3–8 beacons are sufficient to provide efficient peer-location service on 10 000 node Internet-like topologies. Further, our results are 2–5 times more accurate than existing techniques.

We also present results from an implementation of Beaconing over a non-trivial wide-area testbed. In our experiments, Beaconing is able to efficiently (< 3 K Bytes and < 50 packets on average), quickly (< 1 second on average), and accurately (< 20 ms error on average) find nearby peers on the Internet.

1 Introduction

Consider a distributed peer-to-peer application, such as an application-layer reliable multicast service or Gnutella. When a new node joins the application, it often has to find another peer (a *friend*) that is already part of the application. Usually, the goal is to find another application peer that is "near" the new host, i.e. the goal is to find a *close friend*. In this paper, we consider the problem of finding such "close friends" over the Internet. We refer to this as the *peer-finding* problem, and in the rest of this paper, we implicitly assume that the goal is to find near peers.

Efficiently locating "friends" is an important problem for many applications, especially the emerging peer-to-peer distributed applications. For example, in application-layer multicast, finding peers is useful for creating efficient distribution trees; for an application like Gnutella, finding the nearest peer can reduce network load for queries and responses. Efficient solutions for peer finding is also beneficial to legacy (non peer-to-peer) applications. Peer finding schemes can be used to locate near mirrors for file transfers [8], or to locate nearby sources in a content distribution network [1]. Peer-finding schemes can naturally be used to implement application-layer anycasting services [4]. Lastly, the reachability of native multicast groups over the Internet is currently being extended by setting up dynamic unicast tunnels between multicast-enabled regions of the Internet [9, 7]. A solution developed for peer finding can be applied directly to create efficient tunnels. Such a solution would not only be useful for multicast, but also for efficiently deploying new overlay networks [2, 1].

In this paper, we specifically consider the case when the number of possible peers is large and individually probing the whole set to find a near server [6] is not a viable solution. We also consider cases when the set of peers changes rapidly and passive measurement-based techniques [18] incur high error rates. Even under these conditions, the peer-finding problem can be solved relatively easily if we assume network-layer assistance such as native IP multicast, Global Internet Anycast (GIA [15]), or Internet-wide distance maps (IDMaps [13]). The goal of this paper is evaluate different solutions to this problem that can be implemented over the Internet without investing in global infrastructure-level changes or additions (as would be required by both IDMaps and GIA). A special case we focus on is when unicast forwarding is the only available primitive, as is the case on the current Internet. (Even though the protocols have been in place for many years, native multicast reachability remains poor or non-existent over large parts of the Internet [3].) In the absence of all network-layer assistance, efficiently solving the peer-finding problem is challenging.

1.1 Challenges for unicast-only solutions

If a native multicast or anycast service is not available, and an infrastructure-based solution such as IDMaps is not viable, then the peer-finding problem poses several challenges. There are two main problems:

- **Finding the peers:** The identities of the current set of peers is dynamic and unknown. Of course, in some applications, there may be a set of peers who are always present, e.g. the data source in a application layer multicast application. However, knowing the identities of a single (or a small set of) nodes that are always present does not necessarily help in solving the nearest peer problem since there may be other closer nodes which are also participating in the application.

- **Estimating distances:** Given a network address of a peer, it is often not possible to reliable estimate how "far" away it is. There has been recent work [16] in using IP addresses and BGP prefixes to estimate proximity of hosts. However, such information is rather course grained (on the order of AS-AS distances) and does not provide enough resolution for many applications. Hence, some other mechanism has to be used to scalably map addresses to useful distances, and we describe a

[1] See, for example, `http://www.stardust.com/cdn`

new technique that works irrespective of the structure of the addresses.

1.2 Overview of our approach

We introduce a new technique called *Beaconing*. A set of application-level processes in the network act as "beacons" and we find peers using mutual distance measurements to these beacons. Our solution most closely resembles the triangulation [12] and weighted triangulation approaches [11]. Like these approaches, our scheme does not require any changes to the infrastructure and works over only IP unicast; however, as described later in this paper, our overall approach is quite different comprared to the triangulation approaches. In our simulation experiments, using the same number of beacons and the same run-time overhead, our schemes provide 2–5 *times* more accurate results.

1.3 Roadmap

The rest of this paper is organized as follows: in the next section we formally define the near peer-finding problem and discuss several possible solutions, including Beaconing. In Section 3, we discuss issues related to the efficient implementation of the beaconing scheme over the current Internet protocols. In Section 4, we present a comprehensive performance evaluation of each of the peer-finding techniques using both simulations and Internet-based measurements. We discuss related work in Section 5, and conclude in Section 6.

2 Peer Finding: Problem Statement and Solutions

We begin with a formal description of the peer-finding problem. We describe our approach in Section 2.2 and discuss existing techniques in Section 2.3.

2.1 Problem Statement

Formally, the peer finding problem can be modeled as follows:

Assume a set of hosts H arranged in some arbitrary topology T. Let A be the current set of hosts that participate in some application ($A \subseteq H$). Let $n \notin A$ be a host that wishes to join the application. The peer-finding problem finds a node p, s.t. $p \in A$ and $\forall q \in A, dist(p,n) \leq dist(q,n)$, i.e. p is already part of the application and is the closest such host to the new host n.

Clearly, the peer finding problem can be solved according to a number of distance measures, e.g. hop count in the underlying network, application-perceived latency, etc. As we will see different mechanisms will implicitly impose their own measure of distance, e.g. if expanding ring multicast is used to solve the peer-finding problem then the distance measure will end up being the hop count in the underlying network. Some methods may not always find the "nearest" peer. We will quantify the performance of each algorithm using both absolute error and relative error (defined as $\frac{dist(q,n)-dist(p,n)}{dist(p,n)}$). [2]

[2]Note that it is possible to define directed versions of the peer finding problem if the distance function is not symmetric. In all of the techniques we evaluate, it is only possible to efficiently find the node that is closest from n, i.e. the solution finds p, s.t. $\forall q \in A, dist(n \rightarrow p) \leq dist(n \rightarrow q)$. Since we cannot influence how directed distances are handled, we do not explicitly consider this issue again.

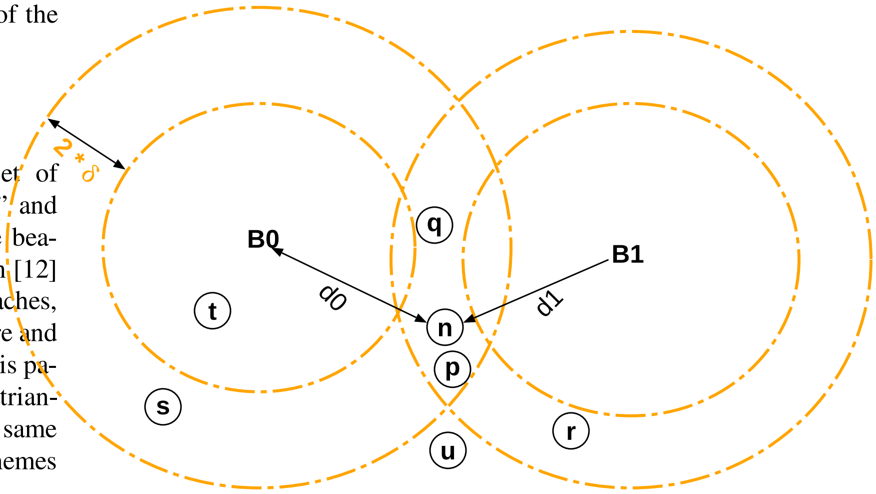

Figure 1: Finding peers using Beaconing

2.2 Finding peers using Beaconing

We designate a set of k hosts ($B_0, \ldots, B_{k-1}$) to be *beacons*. A beacon is a host that measures and collects distance information from hosts in an application. The identity of the beacons could be compiled into the application or it can be dynamic. It is sufficient for each application peer to know the identity of one beacon or the identity of some centralized node which can provide the current set of beacons. In general, beacons are regular Internet hosts and may participate in the application. We next describe an algorithm that will correctly solve the nearest peer-finding problem. Note that the algorithm tacitly assumes that the triangle inequality holds over the distance measure used.

2.2.1 Beaconing algorithm

We use the same terminology as before (A is the set of nodes already in the application, and n is the new peer joining the application). Our description below will refer to Figure 1.[3]

1. Each host h in A periodically measures and reports its own distance (from that beacon) back to the beacon. Each beacon thus maintains a list of its own distance to each host in A. For example, in Figure 1, hosts p, q, ..., u periodically measure their own distances to B_0 and report this value to B_0. They repeat this procedure for all other beacons.

2. Host n measures and sends its own distance d to a single beacon (say B_0).

3. Beacon B_0 sends to n a list containing the identities of all hosts in A within distance $d \pm \delta$. (The beacon compiles this list using the periodic distance information it receives from all the hosts in A.) The parameter δ should be chosen such that the nearest node is within δ hops of n with high probability. Of course, it

[3]The figure is for illustrative purposes only; clearly on the Internet, inter-node distances are not equivalent to 2-D Euclidean distances as shown in the figure.

is possible to underestimate δ and the algorithm will fail in this case. We resolve this issue in Section 2.2.4.

If we assume the triangle inequality holds over the distances, then the list of hosts returned by B_0 list includes all the nodes in A within distance δ of node n, e.g. hosts p and q in Figure 1. Of course, it also includes a number of other nodes , e.g. host s, in A whose distances happen to be in the range $dist(n, B_0) \pm \delta$.

4. Host n repeats this procedure by sending its own distance (in hops) to other beacons b_i and accumulates a number of prospective *near* peers. The hosts in A that are in fact close to n appear in all of these lists.

5. As n communicates with more beacons, the intersection of suitable nodes reduces (to hosts p and q in Figure 1). The procedure is terminated when the resultant set (S) is reduced to some *reasonable* number or when n has communicated with all beacons.

6. The final peer is chosen from S using one of three methods:

   (a) Some node in S is chosen uniformly at random

   (b) Host n measures its distance to all nodes in S and chooses the best one. This technique is only useful if $|S| \ll |A|$. The hosts in S are either very close to n or quite far from n and the granularity of these distance measurements can be relatively coarse.

   (c) The set S is further processed using the information already available at n and some node is chosen as a result of this processing. We next describe a postprocessing heuristic that has performed well in our experiments.

2.2.2 Postprocessing the final set

We compute a new distance measure from n using the following procedure: We assume that the host n is at the origin of a k-dimensional space and we "compute" the distance from n to all nodes in S using the distances returned by the beacons. Let $\mathcal{D}(i)$ be the distance from n to host i in S (assuming n is at the origin of the space). Then:

$$\mathcal{D}(i) = \sum_{j=0}^{k-1} (dist(i, B_j) - dist(n, B_j))^2.$$

We then choose the host i s.t. $\mathcal{D}(i)$ is lowest. This technique is not perfect, i.e. a node i that is not the nearest peer may have the lowest $\mathcal{D}(i)$. For example, some nodes that are relatively far away from n could have the same set of distances to all the beacons due to symmetry in the topology (See Figure 2). In the rest of this paper, we refer to this postprocessing technique as *Vectoring*.

2.2.3 Discussion

Clearly, beaconing is a relatively simple algorithm that can be implemented using only unicast communication. The amount of state at each beacon is $O(|A|)$ and the number of messages exchanged is exactly $k + S$, where k is the number of beacons. Using our post processing techniques, the communication cost of choosing from the final set is reduced to zero, so the number of messages in this case is reduced to k.

There are three interesting issues that immediately have to be considered:

1. Number of beacons: A natural question to ask is how many beacons do we need in order for our techniques to be efficient? It is easy to show that in the 2-D Euclidean case, as long as a suitable δ is chosen, only three beacons are enough to break all symmetry and always provide correct answers. However, no such fixed number is enough in networks with arbitrary topologies. *Our experiments show that relatively few beacons, e.g. 3–8 beacons for a 10 000 node Internet-like topology, are both necessary (to reduce overhead) and sufficient (more beacons don't help).*

2. Placement of beacons: Clearly, if all the beacons are colocated their effectiveness is mitigated. We have experimented with different beacon placements, including k-center placement heuristics. *Our results show that being able to precisely place beacons can be useful, but even beacons placed uniformly at random within stub nodes perform almost as well as the k-center placement.*

3. Distance measurements: The effectiveness of beaconing depends entirely on the robustness of the underlying distance measurements. This is not a problem for simulations since we have complete knowledge of the underlying topology. This issue, however, does have to be confronted for a real implementation. We have developed a cheap and robust latency measurement technique (described in Section 3) that we use for Internet-based experiments.

2.2.4 Correctness

The beaconing algorithm as described will fail if the triangle inequality is violated in the distance measure. This is relatively straightforward to see and an example is shown in Figure 3. In this case, the nearest peer n is not included in the final set and thus cannot be chosen by Beaconing.

The algorithm, however, has a more subtle flaw. Figure 4 shows a case when it fails even if the triangle inequality is preserved by the distance measure. Define the δ-ball around the node n to be the set of nodes within δ hops of n. In general, beaconing fails any time a δ is chosen such that the nearest node is not within the δ-ball of the new node n. Since every node in the final set must be returned by every beacon (because we compute the intersection of the hosts returned by every beacon), our scheme will miss the nearest node if it is not within the δ-ball of node n w.r.t. every beacon.

Note that if the triangle inequality holds, then the error in the beaconing scheme is entirely due to postprocessing *as long as the final node node selected is within the δ-ball around n.* It is possible to repeat the beaconing scheme with increasing values of δ and select the nearest node (if any) within the δ-ball of n. If the triangle inequality holds, then this iterative scheme, at the expense of run-time overhead, will always find the correct node.

2.3 Other Techniques

In the rest of this section, we describe four classes of other techniques to solve the near peer finding problem. We begin with the obvious centralized random selection.

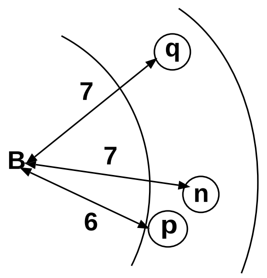

Figure 2: Example where Vectoring fails

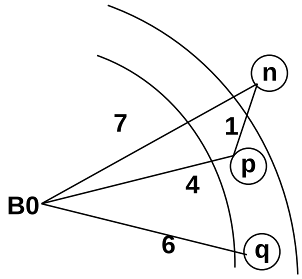

Figure 3: Example where Beaconing fails if the triangle inequality is violated

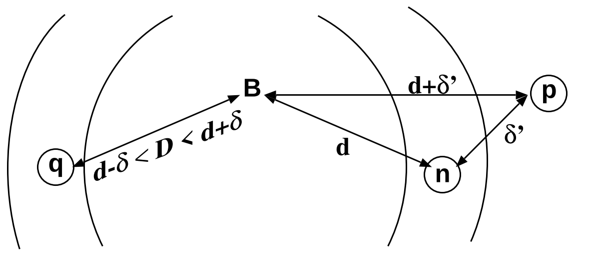

Figure 4: $\delta' > \delta$ and beaconing fails even if the Δ-ineq. holds.

2.3.1 Centralized Random Selection

In this case, all nodes in A register with a well-known node w. When n decides to join the application, w chooses some node $i \in A$ uniformly at random as the "nearest" peer for n. This scheme has extremely low run-time overhead, but also incurs a high error rate.

2.3.2 Triangulation and Weighted Triangulation

The triangulation method, due to Hotz [12], also computes distances using a set of measurement points called beacons.

Given a host i in A and k beacons $B_0, \ldots, B_{k-1}$, the triangulation method described by Hotz [12] defines a n-tuple $D_i = \langle dist(i, B_0), dist(i, B_1), \ldots, dist(n, B_{k-1}) \rangle$. A similar tuple $D_n = \langle dist(n, B_0), dist(n, B_1), \ldots, dist(n, B_{k-1}) \rangle$ is also defined for the new node n. Next, we need the $Avg(i, n)$ function which is defined as

$$Avg(i, n) = \frac{Max(i, n) + Min(i, n)}{2}$$

where the Max and Min functions are defined as:

$$Max(i, n) = \min(|D_i + n_i|) \text{ and } Min(i, n) = \max(|D_i - n_i|)$$

where the $\max$ and $\min$ are the usual arithmetic max. and min. computed componentwise over the k-tuples. The triangulation scheme chooses the peer i with the lowest value of $Avg(i, n)$.

Weighted triangulation [11] uses the same computation but assigns higher distances to backbone links such that after triangulation, the servers which are close to the client have a better chance of being selected. In Internet-based experiments reported in [11], the weighted scheme performs slightly better than Hotz triangulation. However, this scheme requires some way of recognizing when a distance is being measured across backbone links. We only experiment with the pure triangulation scheme in our simulations.

2.3.3 Expanding Ring Multicast and Broadcast

These are two classical techniques that are extremely efficient but require network-level multicast (or broadcast). In expanding ring multicast, all nodes in A join a well-known multicast group. Host n too joins this group and begins sending messages to group with increasing TTL. Since these messages will first be seen by nodes nearest to n (with respect to hop count), they will be the first to respond to n. Host n finds its nearest peer simply by listening on the multicast group for the first response to its peer solicitation messages. This protocol is extremely robust since it does not require any centralized state at any node and is also efficient since it does not involve any application-layer distance computations. The expanding ring broadcast scheme is similar except all nodes in the network receive the messages and only nodes in A respond.

In our performance analysis, we will compare Beaconing to all of the techniques described above (except weighted triangulation). Next we describe two other schemes that can be used to efficiently solve the peer-finding problem. However, both these schemes require new network services (beyond multicast) to be globally deployed. We do not consider these schemes further.

2.3.4 Techniques requiring global service deployment

The peer-finding problem can effectively be solved using Global Internet Anycast [15]. All nodes in A could join an anycast group and n could find a near peer by sending a message to the anycast group. While this scheme could potentially be even more efficient than expanding ring multicast, it requires global anycast support from all participating domains. While there is an effort for standardizing the GIA protocols [14], such services will likely not be globally available in the near future. An Internet-measurement infrastructure such as IDMaps [13] can also be used to solve the peer finding problem. IDMaps are an Internet-wide service that provides distance information between any two nodes in the Internet. IDMaps could be used (along with a centralized directory of current peers) to find near peers or they can even be used in the beaconing protocol to find the best host in the reduced set. Unlike GIA, IDMaps do not require global changes to the network infrastructure; however, IDMaps do require Internet-wide deployment and are not likely to be available in the near future.

3 Implementing Beaconing on the Internet

We have implemented the Beaconing scheme at the application layer and run a number of tests with hosts and beacons on several Internet sites. We were able to access hosts at eighteen different domains, and a snapshot of our test topology is shown in Figure 5. The beaconing code was written in C, and was approximately 500 lines at the hosts and 250 lines at the beacons. The most interesting issue that we had to contend with in our implementation was that of distance measurement.[4]

[4]While we used `traceroute` to create the topology shown in Figure 5, we did not want to rely on `traceroute` in the actual beaconing code. The major reason

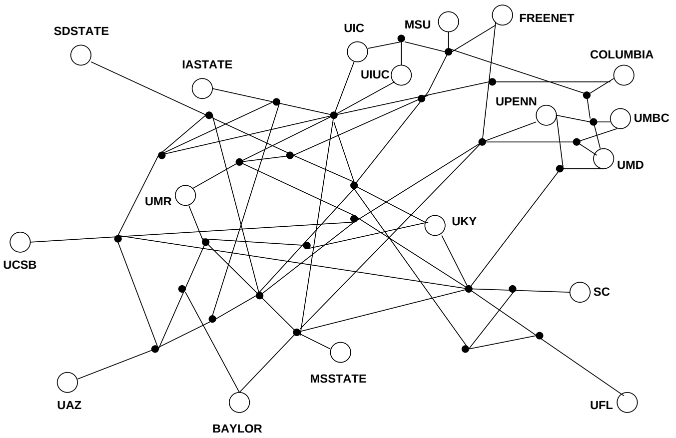

Figure 5: Internet topology used for experiments

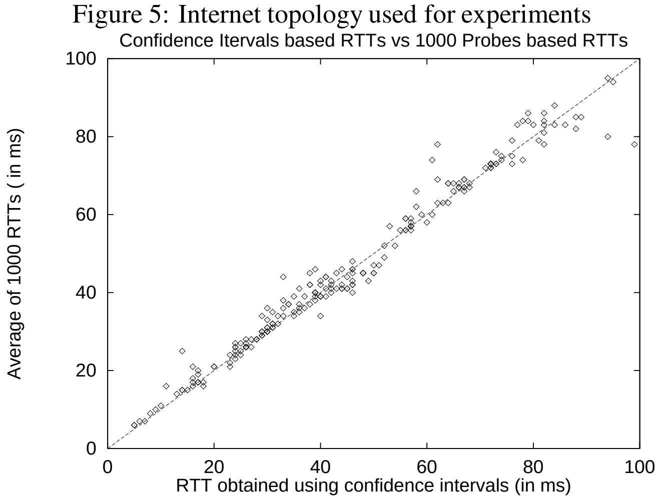

Figure 6: Distance measurement errors

Measuring Distances We used application-layer latency as our primary distance metric. We measure distance by recording the round-trip of 2 byte UDP packets sent between a beacon and application peers. Ideally, we would like to send a few measurement packets and use some function of these measurements (e.g. the mean) of the RTTs as our distance measure. However, we found that the measurements between different host pairs had enough variance that a fixed number of RTTs would certainly be too conservative for some pairs and not enough for others. Instead, we use the following dynamic scheme: we continually measure the 95% confidence intervals of our samples till the intervals reduce to less than 10% of the mean, using at least three samples to compute the RTT.

Figure 6 is a measure of the effectiveness of our distance measurement scheme: each point corresponds to an *actual distance* which is a mean of one thousand probes and a *sampled distance* which was computed using our on-line confidence interval method. The points are distance measurements computed between hosts in our Internet testbed. For each data point, we computed the actual measurement and then analyzed the probe data to determine the value that would have been obtained by the confidence interval computation. In the plot, points on the 45° line correspond to experiments where the sample measurement was exactly the same as the actual measurement and deviations from the 45° line are measurement errors. For the confidence interval scheme, on average we had to use six round-trips before the confidence intervals stabilized. As the plot shows, the confidence interval method works extremely well and compensates for both network-layer congestion and the fact that these experiments were conducted in user-space and were subject to the usual process scheduling and swapping.

was because `traceroute` is not installed on security conscious hosts since it requires superuser privileges. Further, if we use `traceroute`, either it would have to be spawned (several times) by our process, or our code would require root permission to run since traceroute like functionality needs access to raw IP sockets. We wanted a pure application-level solution that could be run by unprivileged applications and users.

Choosing Delta We use the following scheme to choose the value of δ: The new node provides the beacon with an initial delta. If the beacon cannot find any peers for the given delta, it increase the delta until it finds at least one peer. This ensures that all of the beacons return at least one node. In case the intersection of these lists is null, we choose the host that is returned by the maximum number of beacons.

4 Performance Analysis

In this section, we present a set of simulation and Internet-based measurement results about beaconing and compare them to other techniques for peer-finding. We begin with a discussion of our experimental methodology, and show a set of simulation results about Beaconing. In Section 4.3, we compare Beaconing to other techniques including expanding ring searches and triangulations; we conclude this section with experimental results from our implementation.

4.1 Experimental Methodology

We used the Georgia Tech Internet Topology Modeler (GT-ITM [5]) to create our simulation topologies. In our simulations, we used $\sim$10,000 node Transit-Stub (TS) graphs. We experimented on fifty randomly generated TS graphs; each of these graphs had between 700–1000 stub domains. We distributed $\sim$500 application peers uniformly at random within the stub domains. The new node was also placed uniformly at random at some stub domain and we experimented with up to 10 different placements of the new node without changing the set of application peers or the beacons. In all our experiments (except the one that compares beacon placement strategies), we placed beacons in stub nodes chosen uniformly at random.

4.2 Simulation Results

There were three main objectives for our simulations:

- Quantify the accuracy of the beaconing scheme,
- Quantify the effect and overhead of the number of beacons, and
- Understand how beacon placement affects accuracy.

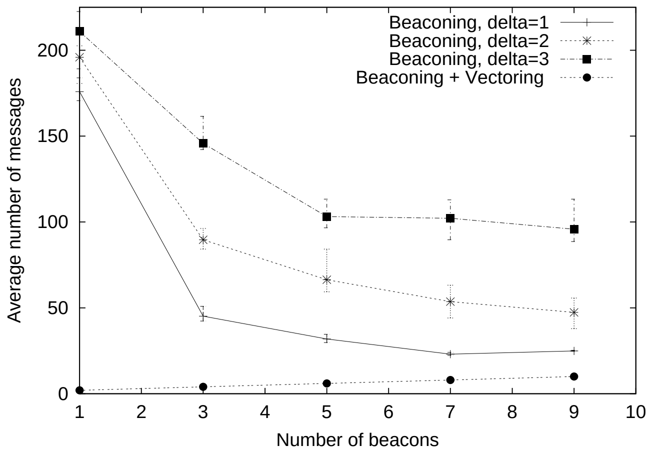

Figure 7: Average message overhead vs. number of beacons

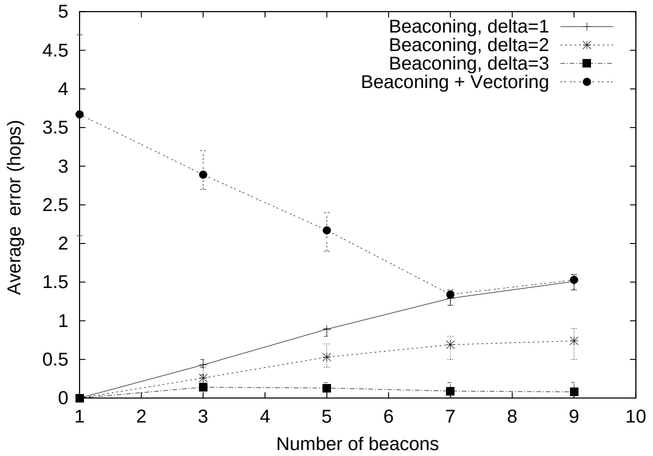

Figure 8: Average error vs. number of beacons

4.2.1 Effect of varying number of beacons and δ

We experimented with different numbers of beacons in our simulations. In Figures 7 and 8, we plot the overhead and accuracy of our schemes as the number of beacons is varied over the same topology. (In Figure 7, each message, regardless of size, is counted once. We report actual byte overheads in Section 4.4). If we use k beacons and S is final set, then for the pure Beaconing scheme, the number of messages is exactly $k + |S|$; in case our postprocessing technique is used, the number of messages is exactly k. As expected, the size of the final set decreases as the number of beacons is increased. In our experiments, about 7 beacons were enough to essentially reduce the size of the final set to its minimum.

As the value of δ is increased, the size of the final set increases. In case all of the hosts in the final set are probed, this directly contributes to higher message overhead. Of course, as more hosts are probed, the resultant accuracy also increases. Thus, the value of δ can be used to trade off some accuracy for message overhead. Using the post-processing technique eliminates the expensive probing of the final set, but almost always incurs a small error. Thus, if an application requires a very high degree of accuracy, it should use a relatively large value for δ and not use the post-processing step. Of course, such applications will pay both in terms of running time and message overhead; however, for most applications, we believe a smaller value of δ is a reasonable compromise.

We have also experimented with a hybrid approach where we probe a fixed number of hosts instead of probing each host. The order in which the hosts are probed is determined by Vectoring. This scheme has the pleasant property of bounded overhead and performs better than pure Vectoring. In our experiments (not plotted), the average error decreases linearly with the number of hosts probed. Specifically, when we experimented with this technique over a using a number of topologies and beacon placements, the average error decreased from ∼2 hops to < 1 hop when we used eight probes instead of just picking the node ranked lowest by Vectoring.

4.2.2 Effect of different beacon placement strategies

Placement Strategy	Final (S) Set size	Message Overhead	Average Error (hops)
Unif. Random	17.5	48.1	1.46
Random Stub Border	17.3	46.6	1.42
K-center	16.6	45.8	1.37

Table 1: Effect of different beacon placement strategies.

Table 1 shows the effect of different beacon placement strategies. We compare placing beacons uniformly at stub nodes to two different techniques. We constrain the beacon placements to only the nodes that are at the borders of stub domains; this is equivalent to the case when beacons are placed at randomly chosen stub gateway routers. This ensures that only one beacon is placed in any single stub domain. In the second case, we place beacons using a k-center heuristic that minimizes the maximum distance from a beacon to any other node. The results shown in Table 1 correspond to scenarios with seven beacons and are averages of 100 different beacon placements with 10 new nodes for each placement. For these experiments, we set the value of δ to be 1: this ensures that the average error from Beaconing is maximized since the resultant set is the smallest. Our results show that being able to place beacons is useful for all metrics; however, even beacons placed uniformly at random perform well compared to the more sophisticated placement techniques we investigated.

4.3 Comparison to other techniques

In this section, we compare Beaconing to Random selection, Expanding Ring Multicast, Expanding Ring Broadcast, and Hotz Triangulation [11]. In Figure 9, we plot the average error for each of these techniques; the average message overhead is shown in Figure 10. For both beaconing and triangulation, we used 7 beacons in these experiments. The points on x-axis for these plots correspond to an experiment index. In each experiment, we fixed the beacon and the A set node placements and varied the placement of the n node uniformly at random. Each point is an average of 10 different placements of the n node and we used 20 different topologies in these experiments.

As expected, random selection incurs the most error and this error is on the order of the diameter of the topology. Obviously, the

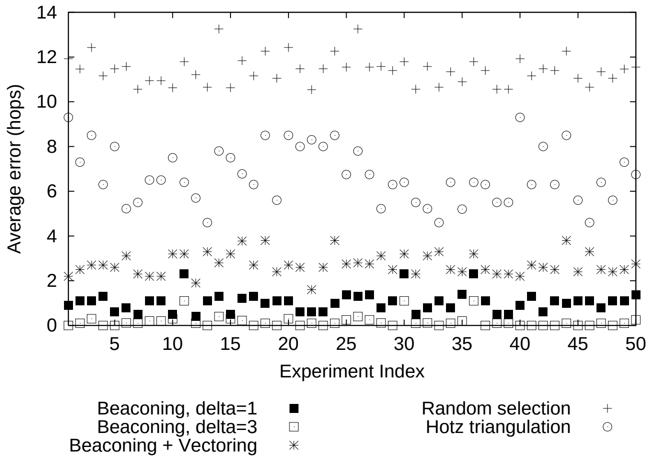

Figure 9: Average Error: Beaconing vs. other techniques

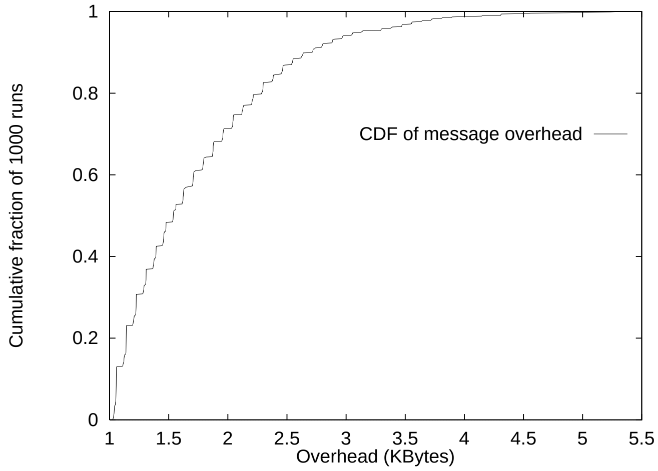

Figure 11: Byte overhead for Beaconing

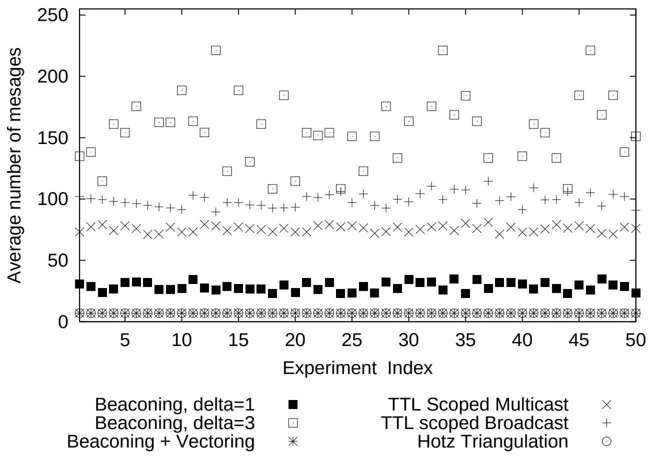

Figure 10: Message Overhead: Beaconing vs. other techniques

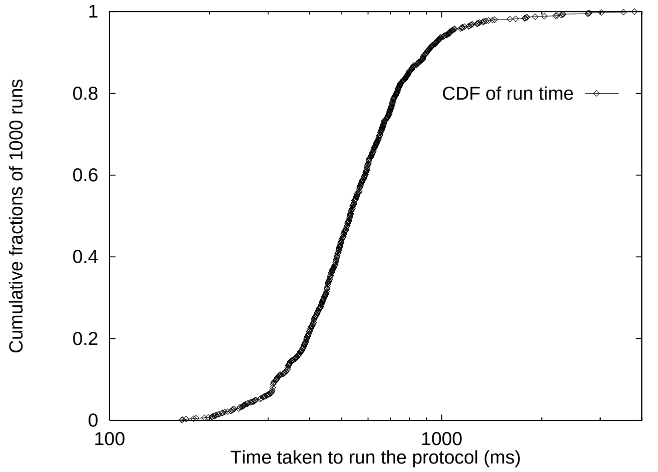

Figure 12: Average time taken for Beaconing

expanding ring searches incur no error and are not shown in Figure 9. Beaconing with $\delta = 3$ is almost as accurate as expanding ring searches but as shown in Figure 10, does incur much higher overheads due to probing. Beaconing with $\delta = 1$ is about 2–5 times accurate than the Triangulation schemes, but does incur about 3–4 times higher overhead. Beaconing with Vectoring is almost as accurate as Beaconing with $\delta = 1$, and has the same overhead as Triangulation. Note that in these topologies, the number of messages sent by the expanding ring searches is more than both beaconing and triangulation. Overall, these experiments show that Beaconing with high δ can be used to provide extremely accurate results at the cost of run-time overhead. Further, Beaconing with Vectoring is almost as effective (average error about two hops) and consumes an order of magnitude less resources. Lastly, we note that we did compute 95% confidence intervals for all of these results but have not included them in the plots to preseve legibility. However, those results do show that the accuracy of Beaconing has significantly lower variance than Triangulation.

4.4 Results from Internet-based Experiments

Beaconing performed extremely well on our wide-area testbed. We ran several thousand experiments over a one month period between 18 different Internet sites (as shown in Figure 5). In these experiments, we used between 3–4 beacons and between 9–14 application peers. We experimented with about one hundred different placements of beacons and clients, and these placements were chosen uniformly at random. Figures 11, 12, 13 show cumulative distributions of the byte overhead, elapsed time, and average error for 1000 randomly chosen experiments. Most of our experiments consume less than 2K bytes (approximately 60 2 byte UDP packets) of network bandwidth per join. This value includes the probes sent for distance measurement and accounts for IP and UDP headers (which account for over 90% of the byte overhead). The vast majority of our experiments took less than 1 second, as measured in application space, to complete. Some experiments (~ 30 out of 1000) did take longer but we believe this is due to packet losses on the wide-area, and local scheduling effects.

Figures 13 and 14 quantify the errors due to Beaconing. There were two sources of errors in our experiments: we used a very small value of δ (as explained in Section 3), and the triangle inequality did not hold between certain sets of sites. We used the small δ to reduce

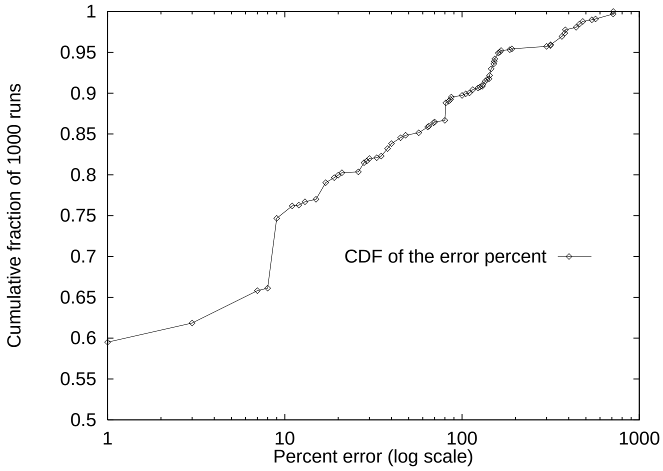

Figure 13: Average error vs. number of beacons

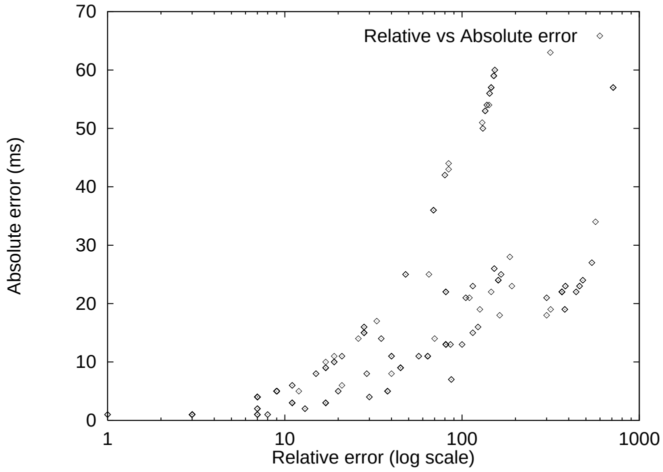

Figure 14: Average error vs. actual error

run-time overhead and these errors can be fixed by using the iterative procedure described in Section 2.2.4. However, the errors due to the violation of the triangle ineq. cannot be handled by Beaconing. As shown in Figure 13, about 60% of our runs were perfect. A small fraction of runs (about 10%) did find a node that was more than twice as far as the closest node. This often happened when closest node was very near and the algorithm selected the second closest node. However, the relative error, plotted in Figure 13, is very high in these cases. We quantify our errors with respect to actual latencies in Figure 14. The x-axis of Figure 14 shows the actual error in milliseconds (i.e. the difference in latency between the actual closest node and the node returned by beaconing). We plot these actual errors against the relative error shown in Figure 13. As is clear from Figure 14, most of high relative errors correspond to relatively low actual errors. Thus, on our wide-area experiments, beaconing is able to find relatively close friends quickly and efficiently.

5 Related Work

Guyton et. al. [11] present a taxonomy for locating peers on the Internet. They classify existing techniques into reactive gathering and proactive gathering categories. Expanding ring searches are classified under the reactive category. Our work can loosely be classified under hop-count probing along with the triangulation-based approaches. We have already discussed Hotz Triangulation [12] and its weighted counterpart [11] in Section 2.3. We note that the number of beacons needed by both types of triangulation, as reported in [11], is an order of magnitude higher than the number of beacons we need.

Network layer anycasting [17, 15] can be used to solve the nearest peer problem by grouping all peers in the same anycast group. Conversely, Beaconing can be used to implement network-layer anycasting. The major difference between previous work in application-layer anycasting [4] and this work is that Beaconing is applicable to very large and dynamic groups. The techniques described in [8, 6] cannot be used to scalably solve the peer-finding problem for large applications.

A completely different approach to finding the nearest peer is to use passive measurements, as described in [18]. If the nearest peer tends to remain relatively static, then passive measurement techniques can be used with great success. Thus, the efficacy of this technique will be a function of application-specific dynamics.

In our work, we did not take the network address structure into account. However, a promising approach is to try to map addresses back to AS numbers and use globally available BGP routing table snapshots to make informed choices about nearby peers. This technique has been used successfully to cluster HTTP clients [16] for the purposes of proxy positioning and server replication. Solving peer-location solely using such clustering is likely to be difficult since AS controllers may not be willing to divulge their topology and all decisions have to be made at AS granularities. However, this technique can be used in conjunction with Beaconing as a postprocessing stop to reduce the size of the final set.

Lastly, a global distance measurement service such as IDMaps [13] can be used to solve the nearest peer problem. The architecture for IDMaps [10] is designed to scale for Internet-wide deployment, while our techniques are specifically designed for per-application use. Some of the distance *inference* techniques developed as part of the IDMaps work, e.g. inferring distances between points given limited set of measurements to Tracers, can be very useful for Beaconing.

6 Conclusions

The problem of efficiently finding nearby application peers has gained in importance with the emergence of wide-area peer-to-peer applications and network services that use replicated servers. In this paper, we have described a new algorithm for finding nearby application-level peers that can be used by all of these applications. Our solution is most closely related to hop-count probing methods such as Triangulation [12]. As such, it can be implemented by arbitrary end-points without investing in any new network-layer infrastructure.

Compared to existing techniques, Beaconing provides between 2–5 times better accuracy in our simulations while using the same amount of resources at run-time. Further, we require an order of magnitude less number of measurement points (beacons) as compared to existing techniques. The run-time overhead of Beaconing can be traded off against accuracy using a single parameter. We have implemented Beaconing and experimented on a non-trivial wide-area testbed. Beaconing is able to efficiently ($<$ 3 K Bytes and $<$ 50 packets on average), quickly ($<$ 1 second on average), and accurately ($<$ 20 ms error on average) find nearby peers on the Internet.

References

[1] 6bone: The IPv6 Backbone. See `http://www.6bone.net`.

[2] Abone: The Active Backbone. See `http://www.isi.edu/abone`.

[3] Kevin Almeroth. A long-term analysis of growth and usage patterns in the multicast backbone (mbone). In *Proceedings of INFOCOM*, Tel Aviv, Israel, March 2000.

[4] Samrat Bhattacharjee, Mostafa Ammar, Ellen Zegura, Viren Shah, and Zongming Fei. Application-Layer Anycasting. In *Proceedings of INFOCOM 97*, 1997.

[5] Kenneth L. Calvert, Matthew B. Doar, and Ellen W. Zegura. Modeling Internet Topology. *IEEE Communications Magazine*, June 1997.

[6] Robert L. Carter and Mark E. Crovella. Server selection using dynamic path characterization in wide-area networks. In *Proceedings of INFOCOMM*, page 1014, Kobe, Japan, April 1997.

[7] Jon Crowcroft and Ken Carlberg. Application level multicast architectural requirements for apex. IETF draft `draft-crowcroft-apex-multicast-00.txt`, feb 2001.

[8] Zongming Fei, Samrat Bhattacharjee, Ellen W. Zegura, and Mostafa Ammar. Finding the Best Server within the Application-Layer Anycasting Architecture. In *In Proceedings of INFOCOM' 98*, 1998.

[9] Ross Finlayson. The udp multicast tunneling protocol. IETF draft `draft-finlayson-umtp-06.txt`, mar 2000.

[10] Paul Francis, Sugih Jamin, Vern Paxson, Lixia Zhang, Daniel Gryniewicz, and Yixin Jin. An Architecture for a Global Internet Host Distance Estimation Service. In *Proceedings of INFOCOM*, New York, March 1999.

[11] James D. Guyton and Michael F. Schwartz. Locating Nearby Copies of Replicated Internet Servers. In *In Proceedings of SIGCOMM*, pages 288–298, 1995.

[12] S. Hotz. *Routing Information Organization To Support Scalable Interdomain Routing with Heterogeneous Path Requirements*. PhD thesis, University of Southern California, 1996.

[13] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. On the placement of internet instrumentation. In *Proceedings of Infocom'00*, Tel Aviv, Israel, March 2000.

[14] D. Katabi and J. Wroclawski. A framework for global ip-anycast (GIA). Internet Draft, Internet Engineering Task Force, June 1999. Work in progress.

[15] D. Katabi and J. Wroclawski. A Framework for Scalable Global IP-Anycast (GIA). In *Proceedings of ACM SIGCOMM 2000*, 2000.

[16] Balachander Krishnamurthy and Jia Wang. On Network-Aware Clustering of Web Clients. In *Proceedings of the SIGCOMM*, August 2000.

[17] C. Partridge, T. Mendez, and W. Milliken. Host anycasting service. Request for Comments 1546, Internet Engineering Task Force, November 1993.

[18] Mark Stemm, Srinivasan Seshan, and Randy H. Katz. A network measurement architecture for adaptive applications. In *Proceedings of INFOCOM*, Tel Aviv, Israel, March 2000.