Locating Nearby Copies of Replicated Internet Servers

James D. Guyton   Michael F. Schwartz

CU-CS-762-95   February 1995

University of Colorado at Boulder

Technical Report CU-CS-762-95
Department of Computer Science
Campus Box 430
University of Colorado
Boulder, Colorado 80309

Locating Nearby Copies of Replicated Internet Servers

James D. Guyton Michael F. Schwartz

February 1995

Technical Report CU-CS-762-95
Department of Computer Science
University of Colorado - Boulder

Abstract

In this paper we consider the problem of choosing among a collection of replicated servers, focusing on the question of how to make choices that segregate client/server traffic according to network topology. We explore the cost and effectiveness of a variety of approaches, ranging from those requiring routing layer support (e.g., anycast) to those that build location databases using application-level probe tools like traceroute. We uncover a number of tradeoffs between effectiveness, network cost, ease of deployment, and portability across different types of networks. We performed our experiments using a simulation parameterized by a topology collected from 7 survey sites across the United States, exploring a global collection of Network Time Protocol servers.

1 Introduction

As an internetwork grows, popular network services often suffer degraded availability and response times. This problem is painfully evident in the Internet, where individual servers and network links are often swamped with tremendous, global service demands. The solution, of course, is caching and replication. Here we focus on a particular aspect of the problem: given a replicated set of servers, how can we distribute service requests in a fashion that minimizes traffic across high-level parts of an internet, such as regional and long-haul links?

The problem is easy for extreme degrees of replication. If there are only a small number of replicas one can use crude approximations, such as mapping Domain names to broad geographic regions, each containing one replica. At the opposite extreme, if a service is replicated into every subnet, one can use broadcast to locate nearby server replicas. In this paper we address the large middle ground. For example, at present there are over 11,000 Network Time Protocol (NTP) servers reachable on the Internet, but the load among these servers is poorly distributed [10]. Other examples of highly replicated Internet services include Harvest Brokers and Caches [4]. We consider a range of techniques, ranging from routing-layer multicast and its derivatives, to application layer techniques using network *probes* ala "ping" [15] and "traceroute" [13] to gather the server location database. We derive experimental cost effectiveness measures for each of the techniques, using a combination of simulation and analysis.

The approaches discussed in this paper apply to any interconnected network that supports some form of distance measurement (discussed in Section 3), and to any set of servers that can be surveyed, either by discovery techniques (e.g., see [10, 18]) or through manual cataloging. Our experimental results focus on the Internet, both because it provides a measurable environment, and because of the practical importance of replicating Internet services.

We begin in Section 2 by considering a range of server location approaches. We discuss distance metrics in Section 3, and network probes in Section 4. We present our experimental methodology in Section 5, and analyze the various approaches in Sections 6- 8. We summarize our findings in Section 9.

2 Server Location Techniques

There are a number of possible ways to locate nearby servers in an internetwork environment, as summarized in Figure 1. The first choice is whether server location information is gathered in *reaction* to client requests for nearby servers (e.g., using a multicast scheme), or whether this information is gathered *proactively*. The next choice is whether support should be provided by the routing layer. Such support can reduce gathering costs, but given the difficulty of achieving widespread router changes we also consider techniques to gather server location information in the application layer. Next, we consider the cost of polling routing tables vs. gathering information via network probes. Given the expense and portability limitations of probe-based gathering (discussed in Section 4), we next consider an approach that attempts to collect partial information from a set of measurement *beacons*, and answer client queries based on a method roughly analogous to triangulation in Global Positioning Systems. Finally, we discuss a hybrid approach that combines the use of route-probing with triangulation.

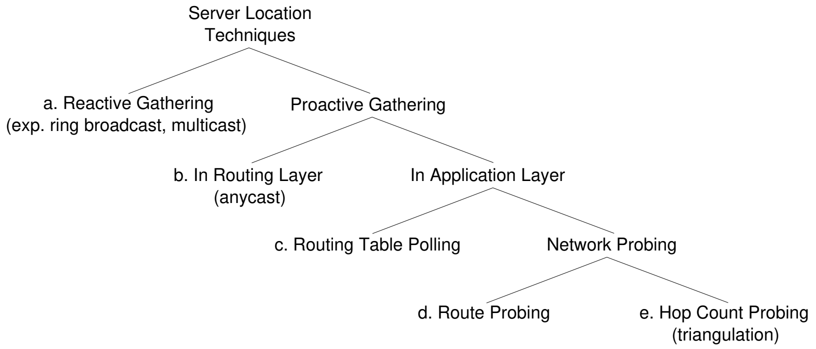

Figure 1: Server Location Techniques Examined in this Paper

Broadcast, Multicast, and Anycast

Some of the earliest wide-area resource location work was done by Boggs. In his thesis he defined a *directed broadcast* mechanism by which a host could send a broadcast to a particular network, and an *expanding ring* broadcast mechanism which allowed broadcasts to increasingly large concentric rings around the broadcasting host [3]. Xerox later incorporated expanding ring broadcasts into their Network Binding Protocol [7]. More recent approaches have considered various forms of multicast routing. That is, given a multicast group joined by all instances of a particular type of server, one can choose the server that responds most quickly to a group multicast message.

Since it is wasteful to transmit a message to an entire group to locate only a few servers, Partridge et al. have proposed an *anycasting* mechanism, which attempts to deliver to one nearby host [16]. While the details of anycasting have yet to be worked out, we observe that anycasting should develop routing information proactively. If it were to work reactively, it would either have to transmit multiple packets at connection request time (which would suffer the same performance problems as using multicast for server location) or it would have to wait for a response from transmitting down one prospective link at a time (which would not necessarily reach a nearby server, and would make it difficult to set meaningful session-layer timeouts).

Anycast is appealing because it can avoid burdening links with repeated requests to gather server distance information. However, a downside is that anycasting assumes all servers provide equal service. [1] It therefore cannot handle server quality differences (such as clock accuracy in NTP servers) without programming policy constraints into routers.

Application Layer Techniques

In contrast to anycast, an application-level location service could allow server quality to be considered as a secondary pruning step, after a handful of nearby servers has been selected from the database. As an additional practical advantage, an application-level service could be implemented without routing support, albeit at higher network cost. The main disadvantage of building the server location database at the application-level is that doing so requires periodic updating. At the routing layer the location database can be constructed incrementally by monitoring routing update traffic and group join/leave requests. Note that one can achieve the same effect at the

[1] [16] specifies that a single server is selected in response to an anycast request, and that any of the servers in the anycast group are equally usable.

application-layer by monitoring routing-layer traffic (e.g., as done by the Fremont system [18]). In this paper we group that approach with proactive gathering at the routing layer - i.e., we distinguish techniques based on whether full routing-layer information is available, not where the actual code resides.

To assess the cost savings from incremental updates it would be necessary to factor in routing and server location change rates. Since we do not have the data needed for this, we relegate that part of the analysis to future work.

Other Related Work

Throughout the 1980s a number of projects considered the problems of locating objects and distributing load in locally distributed environments. While it is impossible to cover them all, we present two representative examples here. Accetta defined a server location protocol intended to be used on top of whatever network protocols are available (e.g., UDP datagrams or broadcasts) [1]. The Eden system used a combination of broadcasts and hint caching to locate objects [2]. From the perspective of the current paper, a key limitation of these systems is that none of them considered network topology; they focused on local area networks.

More recently, a number of systems have been developed that use geographical information to approximate network topology considerations. These include Braun and Claffy's DNS mapping scheme [5], Gwertzman and Margo Seltzer's geographical push-caching scheme [11]. As noted in Section 1, this approach is effective when there are few replicas. Because network topology often does not correspond to geographic proximity, in general this approach may not work well.

A simple strategy for load sharing in wide area environments is random selection among a set of replicas. This approach clearly does not segregate traffic by network region, but is effective in reducing server loading problems. It is being used, for example, by NCSA for their popular WWW server, and by Harvest Brokers [4]. Another simple strategy is to configure routing topologies manually. While labor intensive, this approach is widely used, for example in the MBONE [6] and USENET [14].

The most closely related work to the current paper is Hotz' PhD thesis [12], in which he examines network triangulation as a possible approach to routing and server location. We use some of Hotz' definitions and results in the current paper. Another very relevant effort is Danzig et al.'s *floodd* replication system, which uses network probes to compute logical update topologies over which to send replicated updates [8]. Finally, in [9] Deering proposed an extension to the Internet Control Message Protocol (ICMP) [17], to allow hosts on multicast or broadcast networks to discover their neighboring routers.

3 Distance Metrics

One can make the server location problem arbitrarily complex by generalizing the distance metric to encompass issues such as carrier fees, routing policy, and server quality. In this paper we consider only hop counts and round-trip times, and relegate other network issues to future work. We assume that server quality can be handled by first locating a handful of nearby servers, and then choosing among them based on server quality measures. In turn, this approach presupposes that any network region contains some servers of acceptable quality, and that it is not necessary to find *all* nearby servers.

The first distance metric we considered was round-trip packet transmission time. An important problem with this metric is that backbone links are often faster than local-links, leading to a definition of "nearby" that could favor traversing backbone links instead of localizing traffic. Also, round-trip times are unattractive in the practical example of the Internet because of the combination of poor clock resolution on typical workstation hardware and high variability of round-trip times. These problems led us to use hop-counts as the basic distance metric.

Unfortunately, in the Internet most source-destination pairs are about the same number of hops apart (see Table 1). Thus, hop-counts often do not distinguish between two sites within a regional network vs. two sites separated by wide area links. Given our goal of segregating traffic by topology, the solution is to weight backbone hops more heavily when computing distances. Note that weighting provides a policy "tuning knob", allowing a network administrator to indicate how important it is to localize network traffic.

Sample size	8,098 pairs
Min	1.0 hop
Max	39.0 hops
Mean	17.0 hops
Std Dev.	4.3 hops

Table 1: Internet Hop Count Statistics

We also need to define a way to compute distances when only partial routing knowledge is available (e.g., allowing a distance to be inferred between a client and server given measurements from a measurement beacon to each). For this purpose we use Hotz' definitions [12]. Hotz defines an N-tuple $< s_1, s_2, s_3, ..., s_N >$ as the distance to a node from each of N measurement beacons. He then defines the distance between a server at coordinates S = $< s_1, s_2, s_3, ..., s_N >$ and a client at C = $< c_1, c_2, c_3, ..., c_N >$ with the AVG function:

$$AVG(S,C) = (MAX(S,C) + MIN(S,C))/2$$

where MIN and MAX are defined as:

$$MIN(S,C) = max(|s_1 - c_1|, |s_2 - c_2|, |s_3 - c_3|, ..., |s_N - c_N|)$$

$$MAX(S,C) = min(s_1 + c_1, s_2 + c_2, s_3 + c_3, ..., s_N + c_N),$$

and min and max are the usual arithmetic minimum and maximum.

The rationale for this formula is developed in [12]. The intuition is illustrated in Figure 2. As measured from B, S could be located either at s' or at s'', and C could be either at c' or c''. The distance between S and C is therefore bound by $|C - S|$ and $C + S$. Generalizing to N beacons produces the MIN and MAX functions above. While these provide upper and lower distance bounds, Hotz showed that the AVG function generally produces a better result than either MIN or MAX.

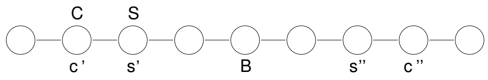

Figure 2: Motivation for Hotz' Distance Metric

4 The Nature of Network "Probes"

There are two different types of probes needed for the techniques listed in Figure 1: *routing probes* and *hop-count probes*. The former can be done using a technique such as traceroute, while the latter is less cost costly and more likely to be available on networks other than the Internet. [2] These advantages of hop-count probes are the primary reason we consider triangulation (technique (e) of Figure 1).

Traceroute works by transmitting a sequence of packets with increasing "Time-To-Live" (TTL) values [3] , and observing the sequence of hosts that respond with ICMP "Time Exceeded" packets. For hop-count probing we optimize this approach by sequencing TTL values according to a binary search algorithm, and seeding the counter with a "guess" corresponding to the average number of hops between nodes (17 in the current Internet, as noted in Table 1).

Because the cost of performing hop-count probes is needed in the analysis for this paper, we computed the average cost of performing such probes on a set of Internet hosts using the above algorithm, and counting the number of iterations needed to converge. Table 2 provides the results. We use the mean value of 5.2 later in the paper.

Sample size	8,098 destinations
Min	3.0 iterations
Max	18.0 iterations
Mean	5.2 iterations
Std Dev.	1.5 iterations

Table 2: Measured Hop-Count Probe Costs

5 Experimental Methodology

To compare the performance of the various server location techniques, we constructed a simulator that inputs the topology of a simulated network of clients and replicated servers. To provide a realistic topology, we began with a list of 11,000 Network Time Protocol (NTP) servers on the Internet, as discovered by an earlier survey [10]. We coalesced this graph to one server per IP network, to reduce the Internet load needed to determine the topology, and to reduce the scale of the simulation. This reduced the list to about 1,200 hosts yet provides a very good approximation, because we are primarily concerned here with reducing wide area traffic. Also, the number of hops

[2] For example, given the ability to set TTLs at the sending side, one can use a simple timeout scheme to determine hop counts.

[3] "Time-To-Live" is a misnomer - this field should really be called "Hops-To-Live". We continue to call it TTL in this paper because the term is widely used in the Internet.

to each node on the Internet turn out to be pretty similar: we measured a mean difference of 0.68 and a standard deviation of 2.07.

Next, we built a graph representing the Internet connectivity among the NTP servers, by performing routing probes from each of the seven test sites listed in Table 3 to each of the 1,200 NTP networks.

Host	**Location**
aja.rand.org	Santa Monica, CA
aloe.cs.arizona.edu	Tucson, AZ
baldwin.udel.edu	Newark, DE
burton.cs.colorado.edu	Boulder, CO
catarina.usc.edu	Los Angeles, CA
largo.lbl.gov	Berkeley, CA
mercury.lcs.mit.edu	Cambridge, MA

Table 3: Locations of Experimental Measurement Sites

Because this graph construction is intended to model general replicated services, throughout the remainder of this paper we refer to the servers in the graph generically as "replicated servers" rather than as "NTP servers".

Our simulation allows us to measure *packet-hops* needed to build server location databases for each of the techniques illustrated in Figure 1. To validate the simulation, Figure 3 plots packet-hop histograms produced by running the original 8,400 routing probes (among the 7 measurement beacons and 1,200 NTP networks), and as computed by the simulation's graph traversal. The computed values are somewhat smaller than the measured values because the simulation does not account for real-world factors (such as dynamic routing around saturated links, and the "topology of politics") that sometimes cause the Internet to use suboptimal routes. This reduction is consistent across our simulations and hence does not effect result ranking. Note also that we do not take packet drop rates into account for any of the server location techniques. This means that in practice the actual network load will be somewhat higher than what we compute here, but the relative costs of the various algorithms will be as we computed them here.

Augmenting the simulator with some mathematical analysis of the simulated environment, in the remainder of this paper we generate cost and effectiveness measures for each of the techniques listed at the labeled leaves of Figure 1. In some cases it is difficult to compare the techniques because, for example, it makes sense to use different numbers of beacons for collecting the needed information in route probing than in hop-count probing. We address this difficulty by comparing costs and effectiveness for *likely* scenarios of how each system would be deployed. We summarize the results at the end of the paper in Table 5.

6 Gathering Location Data with Router Support

Routing layer support is required for techniques (b) and (c) in Figure 1. We consider each below.

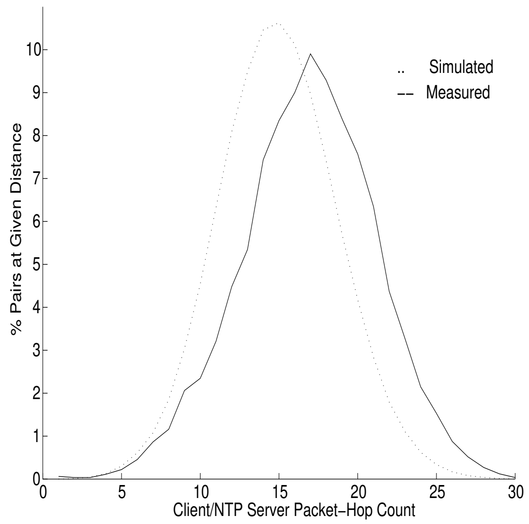

Figure 3: Validating Graph-Based Simulation

Anycast

To analyze the cost-effectiveness of anycasting, we postulate the following mechanisms. When a new replicated server joins an anycast group, the local router advertises the existence of the service as part of its normal routing exchanges with the neighboring routers. If the local router notices that a replicated server has stopped advertising, this negative information would also be passed along to the neighboring routers.

The neighboring routers each examine a distance metric for the advertised anycast server, update their tables based on this metric, and in turn pass along the updated information to their neighbors. Each router retains knowledge of at least the route to the nearest anycasting servers, and possibly a small list of alternative servers in case it is notified that the previously nearest server is down. In this way each router in the Internet would have enough knowledge to route anycast packets to the nearest server, but would not need to keep a database of all the anycast servers in the network. The number of servers that are listed provides a tradeoff between router memory requirements and anycast success probability.

At replicated server location time, a client sends a connection request to the anycast group, and if it is not a service on the local subnet, the local router forwards the packet towards the nearest replicated server based on the current routing tables.

This cooperation between routers to inform the Internet of anycast servers would work in a manner very similar to that of advertising routes. While the detailed algorithm and a number of practical issues (such as security and the potential for broadcast storms) still need to be worked out, we expect that an efficient implementation of this would work very well.

Compared with the other replicated server discovery techniques presented in this paper, the additional traffic to support anycasting could be provided at essentially no measurable cost.

Polling Routing Tables

The cost of polling routing tables can be computed by observing that we can build a connectivity graph from a measurement beacon to one server by retrieving the local routing table, determining the next hop along the path, retrieving the routing table for that router, and so on, until we reach the destination. We can extend this algorithm to discover routes to all servers by iterating over the servers, taking care not to retrieve a routing table that has already been retrieved. Therefore, the cost of discovering the routes to all replicated servers is simply the cost of retrieving all of the routing tables in the set of paths to all of the servers.

To compare costs with those simulated in the other sections of this paper, we assume a simple TCP exchange to retrieve each table will require approximately 10 packets to retrieve an average size routing table. [4] Since on average each router is 17 hops away (see Table 1), the cost is 170 packets per router.

To build the full routing table for our 1,200 host experiment, we discovered 5,563 routers in our Internet survey. Therefore, the cost to build the entire server location routing database would be 170 * 5,563 = 945,710 packets.

7 Gathering Location Data Using Routing Probes

In this section we analyze the costs of gathering a full connectivity graph for server location without the ability to poll routing tables (technique (d) of Figure 1). We begin with a set of measurement servers, which explore the routes to each of the replicated servers. When a client asks one of the measurement servers for a list of nearby replicated servers, a measurement server explores the route back to the client and adds that information to its connectivity database. It then searches the database for servers near the client.

To measure this technique's effectiveness at discovering enough of the actual connectivity graph to find the correct nearby servers, we began with the 1,200 replicated server connectivity graph (described in Section 5), and computed how many servers a client would have to examine before finding at least 3 of the 10 nearest replicated servers. While this represents a likely-to-be-useful case, we also computed how many servers a client would have to examine before finding at least 9 of the 10 nearest replicated servers, as a "stress test". Figure 4 plots each of these results, using graphs built from one, two and three survey sites. Each survey adds more information to the connectivity graph, allowing for a more accurate determination of what is close to a client.

For each plot the x-axis is the number of hosts the client examines, and the y-axis reports the probability the client is able to find at least the given proportion of closest hosts. For example, using 1 survey, a client must probe 3 of the replicated servers to have a .95 probability of finding at least 3 of the nearest 10 servers.

Figure 5 shows why just using just one survey site is so effective: In terms of both nodes in the graph and links between nodes, a single survey generates most of the needed information, and more routing probes (to the same 1,200 replicated servers) add little information to the connectivity graph.

[4] Taking the low-level connection setup, data packets, and acknowledgements into account, we estimate that approximately 1,365 routes can be transmitted in 10 packets. We do not have measurements indicating average routing table size in the Internet, but rather offer this as a rough order-of-magnitude computation for the cost of collecting the needed information.

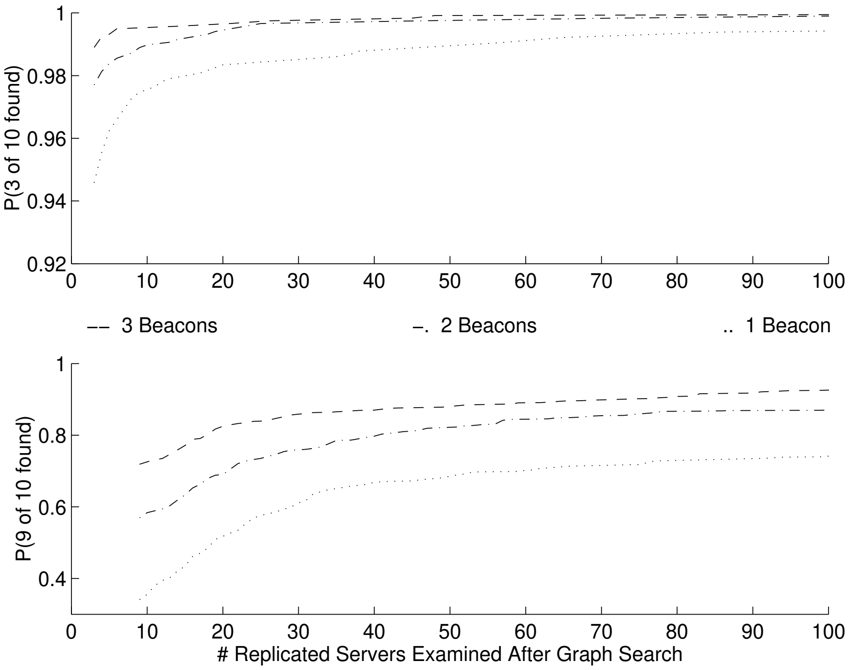

Figure 4: Effectiveness of Gathering Location Data Using Routing Probes

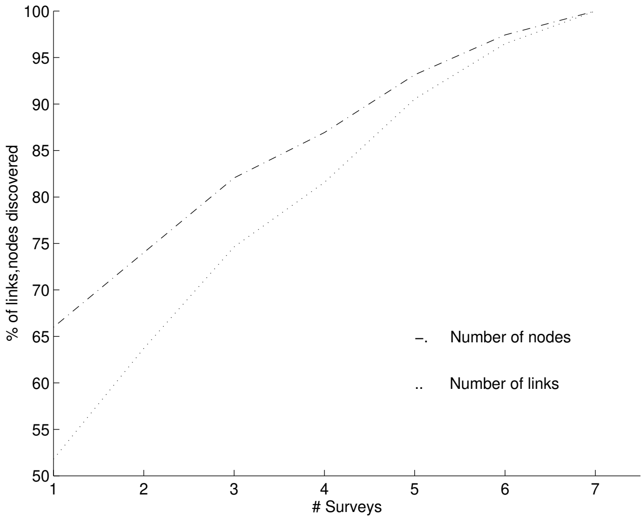

Figure 5: Node and Link Count Growth for Single Survey Site

7.1 Cost Analysis of Using Routing Probes

The cost of this technique for N survey sites is:

$$TotalCost = N * C_{RP} + C_q,$$

where C_{RP} is the cost of a routing probe survey, and C_q is the cost of a client query. We can compute C_{RP} as follows:

$$C_{RP} = R * C_{rp},$$

where R is the number of replicated servers and C_{rp} is the per-server routing probe cost. C_{rp} involves a round-trip packet cycling through each of the Time-To-Live values between the sender and the destination. For hosts H hops apart we have:

$$C_{rp} = \sum_{i=1}^{H} 2i = H(H+1) \text{ packets}$$

Again assuming an average of 17 hops between hosts, we have an average value for C_{rp} of 306 packets. Therefore, for our experimental set of 1,200 replicated servers and one survey site, we have:

$$TotalCost = N * R * 306 + C_q = 367,200 + C_q$$

Using the "best 3 of 10" test, the technique works so well that the client can take the results of the query on faith, even when there is only one survey site. So the cost of a query is just the cost of a routing probe to the client from a single measurement server, which is 306 packets.

So the total cost of the technique for q clients making queries against the location database (before it needs re-gathering) is approximately:

$$367,200 + 306q$$

8 Gathering Location Data Using Hop-Count Probes (Triangulation)

In this section we analyze the costs of gathering a subset of server location data using triangulation (technique (e) of Figure 1).

The triangulation technique works as follows. Starting with a list of replicated servers and N *beacon* servers distributed around the network, each beacon first measures its distance to each server. Each beacon then sends its measurements to a *triangulation server*, which combines the per-replicated server measurements to produce a measurement coordinate for each replicated server in triangulation space (as discussed in Section 3). As a practical matter, the beacons could circulate measurements among each other and produce the coordinates themselves, and serve the joint role of measurement beacons plus replicated triangulation servers. Below we discuss separate servers for the sake of clarity and simplicity.

When a client wishes to locate a nearby server, each beacon measures its distance to the client, and the triangulation server collects these measurements to determine a coordinate for the client. Based on the client's coordinates and the database of replicated server coordinates, the triangulation server then generates a list of servers whose coordinates are closest to that of the client, using the AVG distance metric defined in Section 3.

Unfortunately, the triangulation coordinate space is sensitive to beacon placement. Figure 6 provides an example, with distances and coordinates given in Table 4. The coordinates imply that S_1 and S_2 are at the same location, but in fact they are 8 hops apart. Similarly, C_1 and C_2 are 6 hops apart, but appear co-located. Because there is no beacon on the "horizontal axis", it is impossible to distinguish the locations of S_1 and S_2. As an aside, the problem illustrated in Figure 6 does not occur in physical triangulation because the coordinate space for physical triangulation is independent of beacon placement, allowing both distances and *angles* to be computed. All that matters is that the angles between beacons and triangulated objects be sufficiently large.

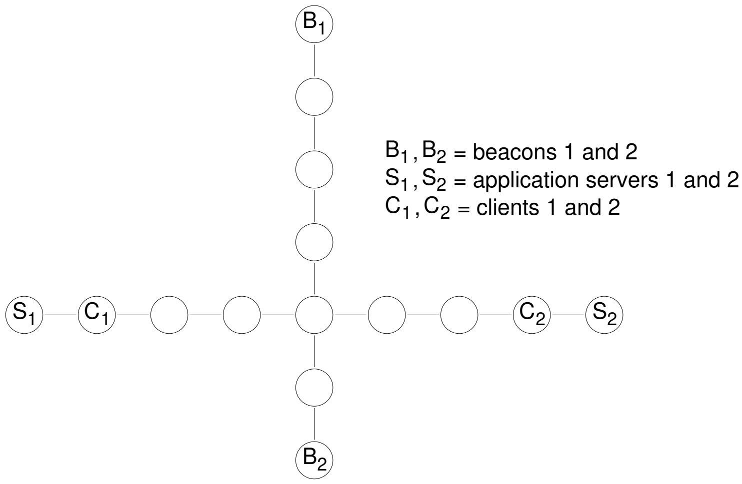

Figure 6: Why Simple Triangulation Works Poorly

Client or Server	Distance to B_1	Distance to B_2	Coordinates
S_1	8	6	<8,6>
S_2	8	6	<8,6>
C_1	7	5	<7,5>
C_2	7	5	<7,5>

Table 4: Distances and Coordinates for Example Triangulation Scenario

But even with unambiguous locations for servers and clients, triangulation cannot accurately compute the distance between two measured points. What it can do is place bounds on the minimum and maximum distances between two measured points [12]. These bounds can be used as an initial filter, with clients selecting among the resulting subset by measuring their own distances to each of the remaining replicated servers. We next explore the effectiveness of this filtering approach.

8.1 Effectiveness of Triangulation

Our first triangulation experiment assesses how many replicated servers a client must consider among those output by the triangulation filtering step, to find a given proportion of the nearest ones. We ran simulations of beacons randomly placed on the graph, with 1,180 clients trying to locate nearby replicated servers in each case. The results are shown in Figure 7. For each plot the x-axis is the number of triangulation-winnowed hosts the client examined, and the y-axis reports the probability a client is able to find at least the given proportion of closest hosts. For example, using 50 beacons a client must probe 24 post-triangulation servers to have a .75 probability of finding at least 3 of the nearest 10 servers. The top plot also shows the results of picking hosts at random, for comparison purposes. (We tried random probing for both plots, but the results never climb above probability 0 for the 9-of-10 plot.)

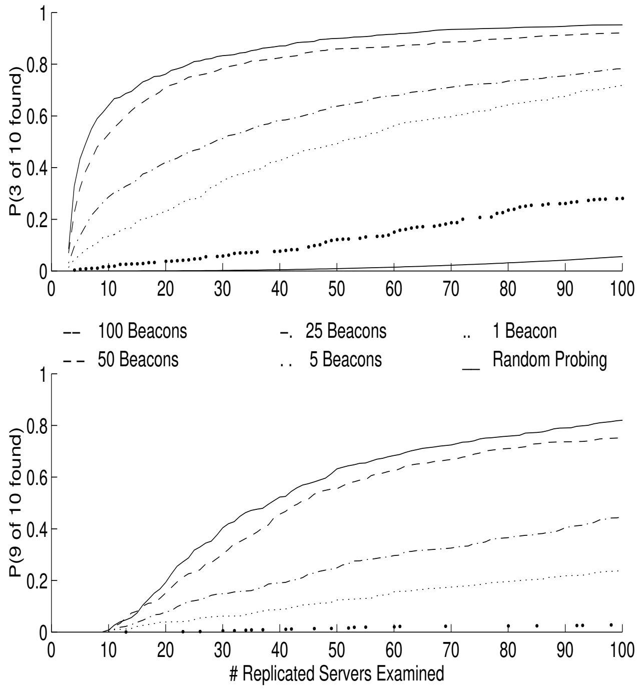

Figure 7: Effectiveness of Triangulation

As an experiment, we also tried weighting backbone links. As noted in Section 3, the primary reason for weighting is to reduce backbone traffic. However, there's another reason as well: if we can discourage backbone traffic, we may also make triangulation more likely to locate nearby servers, and hence increase its effectiveness. We used a simple heuristic to determine weights, namely, determining which links are shared by the most routes. We then weighted these routes 10 times as high as the other routes. The results are shown in Figure 8.

As can be seen, weighted triangulation outperforms unweighted triangulation by a moderate amount. For example, for the .75 probability 3-of-10 case noted in the analysis of pure triangulation above, the client would need to examine only 11 rather than 24 of the replicated servers output by the triangulation step. Note that this approach represents a hybrid between the approaches of Section 7 and the current section, since it requires route probes, not just hop-count probes. Also,

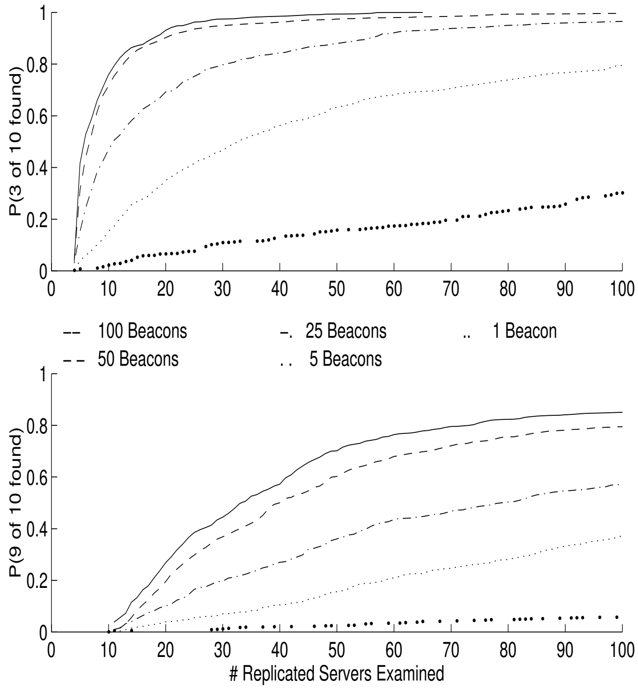

Figure 8: Effectiveness of Weighted Triangulation

while weighted triangulation is still more costly than gathering location data using routing probes, it is more portable, so may still be of value.

8.2 Cost Analysis of Triangulation

Triangulation incurs costs to gather the triangulation database, costs for each of the beacons to measure their distances to the client, and costs for the client to measure its distance to to each of the hosts output by the triangulation step.

To build the triangulation database, each of B beacons must probe each of N replicated servers. The cost of a probe is $P * H$, where P is the average number of test probes needed per measurement, and H is the average number of round trip hops per test probe. This gives $B * N * P * H$ as the total cost to build the database.

Since hosts are an average of 17 hops apart, $H = 34$ and $P = 5$. Hence, with our sample set of 1,200 replicated servers, the cost of building the triangulation database is approximately B * 204,000 packets.

The cost of a client request is the cost of the beacons to measure to the client plus the cost of the client to measure to each of the replicated servers output by the triangulation step until the desired number of nearby servers have been located.

The cost of the beacons to measure to a client is just $B * P * H$. With the current values for P and H, we get $B * 170$ packets.

Given that non-local hosts are 17 hops away on average, the cost to the client of probing M hosts is $M * P * H$ or $M * 170$. For example, using the data from Figure 7 for a system with 50 beacons, a client wishing to locate 3 of the nearest 10 replicated servers with probability of .5 would have to examine 9 post-triangulation replicate servers, with is $9 * P * H$ packets. Assuming the above values of P and H, this would require 1,530 packets.

Combining the costs of building and using the triangulation database, the full cost of this technique for a 50 beacon system and 75% success for finding 3 of the nearest 10 replicated servers (a likely scenario), we have 10.2 million packets to build the triangulation database, 8,500 packets to measure distances from the beacons to the client, and 4,080 packets for the client to probe the 24 post-triangulation servers. Since the first two quantities can be reused until the database gets stale and must be regathered, the overall cost of the approach is approximately

$$10.2 \text{ million packets} + 4{,}080q,$$

where q is the number of queries serviced before the database is regathered.

9 Summary

In this paper we have analyzed techniques for locating nearby server replicas in an internetwork, with the overall goal of minimizing long-haul traffic. The problem is of particular relevance for the blossoming collection of upper layer Internet services such as NTP, WWW, and Archie.

In our analysis we struck a balance between considering general internetworks, and focusing on a solution to the very real problems of the Internet. We parameterized a client/server interconnection graph from data about the locations of NTP servers, and built a simulation that allowed us to experiment with a number of possible approaches.

Table 5 summarizes our findings. This table is a bit unusual because it lists *absolute* packet counts in the efficiency column. These counts are to be interpreted as experimental outcomes for our particular simulations; they provide rough order-of-magnitude comparisons between the techniques. Another point to note in this table is that in some cases it is difficult to compare the techniques because, for example, it makes sense to use different numbers of beacons for collecting the needed information using the different techniques. To allow the techniques to be compared, Table 5 therefore reports values for *likely* scenarios of how each system would be deployed (e.g., the 75% and 95% values in the "Effectiveness" column determined the corresponding values in the "Cost" column).

Approach	Exper. Cost (packets)	Effectiveness	Comments
a. Reactive Gathering	-	-	cannot guarantee cost limits
b. Anycast	0 (piggybacked on routing protocol)	hard to guarantee nearness	practical deployment hurdles; doesn't support QoS
c. Routing Table Polling	$1,000,000 + q$	100%	handles QoS; fewer practical hurdles but still non-trivial
d. Route Probing	$367,200 + 306q$	95% to find 3 of nearest 10	very deployable; reasonable cost
e. Hop-Count Probing (triangulation)	$10,000,000 + 4,080q$	75% to find 3 of nearest 10	most portable method

Table 5: Summary of Salient Characteristics of Techniques Analyzed in this Paper

Several points are worth noting in this table. First, from an efficiency perspective anycast is the clear winner. However, the combination of practical deployment issues and limits for supporting service quality choices mean it is worth considering the other techniques. Also, the techniques that lack routing layer support must periodically rebuild their databases, as their server/topology information ages. q is the number of client queries that are performed between database re-gatherings. The cost of building the location database is amortized across client queries until the database becomes stale and must be regathered.

We originally considered triangulation because intuitively it seemed the technique would be cost effective, since it is less costly to collect hop-count information than routing information. Unfortunately, it turns out that hop counts are not sufficient to provide good server location choices, and hence many triangulation beacons must be deployed and clients must test the results of a triangulation query – at great network expense.

We also examined a hybrid solution (not shown in the table), involving weighting triangulation links to discourage backbone traffic. This technique worked moderately better than triangulation, but still is no match for collecting full routing information.

One last note concerns the magnitudes in the table. At first blush generating 10 million packets to build a table sounds pretty excessive. However, this expense can be amortized among a large, global set of clients. If this is done, 10 million packets really is not a large amount of traffic, on the scale of the global Internet. Given the portability of hop-count probing, this technique is at least worth consideration.

Future work

Given data about the rates of change of Internet routes and server locations, a worthwhile area of future exploration would be to assess the cost savings from incremental updates (ala anycast) over building the server location database at the application level. Another area of future work is extending the distance metric to support more complex networking considerations, such as carrier fees and routing policy. We also intend to run experiments to determine the cost effectiveness under more careful beacon placement. On a related note, we will explore triangulation weighting algorithms more fully, in the hopes of enhancing cost effectiveness.

Finally, once we finish working through all of these experiments, we would like to build and deploy a server location system for the Internet.

Acknowledgements

This work was supported in part by the Advanced Research Projects Agency under contract number DABT63-93-C-0052, the National Science Foundation under grant number NCR-9204853, and an equipment grant from Sun Microsystems' Collaborative Research Program.

The information contained in this paper does not necessarily reflect the position or the policy of the U.S. Government or other sponsors of this research. No official endorsement should be inferred.

We thank Peter Danzig, Udi Manber, Dave Mills, Vern Paxson, Terry West, and John Wroclawski for allowing us to run measurement software at their sites.

References

[1] Michael Accetta. RFC 887: Resource location protocol, December 1983.

[2] Guy T. Almes, Andrew P. Black, Edward D. Lazowska, and Jerre D. Noe. The Eden system: A technical review. *IEEE Transactions on Software Engineering*, SE-11(1):43–59, January 1985.

[3] David R. Boggs. Internet broadcasting, January 1982. Ph.D. Thesis, available as Technical Report CSL-83-3, Xerox Palo Alto Research Center, October 1983.

[4] C. Mic Bowman, Peter B. Danzig, Darren R. Hardy, Udi Manber, and Michael F. Schwartz. Harvest: A scalable, customizable discovery and access system. Technical report, Department of Computer Science, University of Colorado, Boulder, Colorado, August 1994. CU-CS-732-94.

[5] Hans-Werner Braun and Kimberly C. Claffy. An experimental means of providing geographically oriented responses relative to the source of domain name server queries. Technical report, San Diego Supercomputing Center, April 1994.

[6] Steve Casner. Frequently Asked Questions (FAQ) on the multicast backbone (MBONE), December 1994. Available from ftp://ftp.isi.edu/mbone/faq.txt.

[7] Xerox Corporation. Network Binding Protocol. Technical report, Xerox Corporation, Network Systems Institute, Palo Alto, California, June 1986.

[8] Peter Danzig, Katia Obraczka, Dante DeLucia, and Naveed Alam. Massively replicating services in autonomously managed wide-area internetworks. Technical report, University of Southern California, January 1994. USC-CS-93-541.

[9] Steven E. Deering. RFC 1256: ICMP router discovery messages, September 1991.

[10] James D. Guyton and Michael F. Schwartz. Experiences with a survey tool for discovering Network Time Protocol servers. *Proceedings of the USENIX Summer Conference*, pages 257–265, June 1994.

[11] James Gwertzman and Margo Seltzer. The case for geographical push-caching. Technical report, Harvard University, 1994.

[12] Steven Michael Hotz. Routing information organization to support scalable interdomain routing with heterogeneous path requirements. Technical report, Computer Science Department, University of Southern California, Los Angeles, California, September 1994 (Draft). PhD Thesis.

[13] Van Jacobsen. Traceroute software, December 1988. Available from ftp://ftp.ee.lbl.gov/pub-/traceroute.tar.Z.

[14] B. Kantor and P. Lapsley. RFC 977: Network News Transfer Protocol - a proposed standard for the stream-based transmission of news, February 1986.

[15] Michael Muuss. Ping software, December 1983. Avaliable from ftp://uunet.uu.net/bsd_sources-/src/ping.

[16] Craig Partridge, Trevor Mendez, and Walter Milliken. RFC 1546: Host anycasting service, November 1993.

[17] Jon Postel. RFC 792: Internet control message protocol, September 1981.

[18] David C. M. Wood, Sean S. Coleman, and Michael F. Schwartz. Fremont: A system for discovering network characteristics and problems. *Proceedings of the USENIX Winter Conference*, pages 335–348, January 1993.