Dynamic Server Selection using Bandwidth Probing in Wide-Area Networks

Robert L. Carter and Mark E. Crovella
Computer Science Department, Boston University
111 Cummington St., Boston MA 02215
{carter,crovella}@cs.bu.edu

BU-CS-96-007
March 18, 1996

Abstract

Replication is a commonly proposed solution to problems of scale associated with distributed services. However, when a service is replicated, each client must be assigned a server. Prior work has generally assumed that assignment to be static. In contrast, we propose dynamic server selection*, and show that it enables* application-level congestion avoidance.

To make dynamic server selection practical, we demonstrate the use of three tools. In addition to direct measurements of round-trip latency, we introduce and validate two new tools: bprobe*, which estimates the maximum possible bandwidth along a given path; and* cprobe*, which estimates the current congestion along a path.*

Using these tools we demonstrate dynamic server selection and compare it to previous static approaches. We show that dynamic server selection consistently outperforms static policies by as much as 50%. Furthermore, we demonstrate the importance of each of our tools in performing dynamic server selection.

1 Introduction

The increasing popularity of distributed information services like the World Wide Web has resulted in a number of problems of scale. Three scaling impediments to such distributed information services are excessive server load due to popular documents, wasted network bandwidth due to redundant document transfer, and excessive latency in delivering documents to the client due to the potential for transfers over slow paths. A technique that promises to address all of these problems is service (or document) replication. However, when a service is replicated, clients face the additional task of finding the best provider of that service.

In many cases, clients may know in advance which service providers are best for them. Such a static server selection scheme is used, *e.g.,* in the distribution of network news using NNTP. However, static server selection is inappropriate in a number of cases. For example: 1) The authors in [6] employ a dynamic protocol to find the best server for a document: the client probes a set of servers who may have a replica of the document, as well as the document's home server, and chooses the first to reply as the source of the document. 2) A mobile client using a replicated service will want to find the best server to serve the client's requests; the choice will depend on the changing location of the client. 3) Finally, our current interest is in replicating WWW documents. In this context, dynamic server selection has the potential to reduce the elapsed time between document request and document arrival, which we will call the response time. In addition, as we will show, a dynamic server selection mechanism enables a very flexible, simple policy for document replication.

In this paper we report on tools and techniques for selecting good service providers without assuming knowledge of server location or network topology. Our only assumption is that the client can obtain a list of addresses of servers that provide the required service.

In our initial experiments we consider two principal metrics for measuring distance in the Internet — hops, and round-trip latency — and study their use as the basis for a server selection decision. Surprisingly, we show that these two metrics yield very different results in practice. We then present evidence that dynamic server selection polices based on instantaneous measurements of round-trip latency provide considerable reduction in response time compared to static policies (which in this case are based on either geographical location of client and server or distance measured using network hops).

However, round-trip latency alone does not capture all the information one would want about the quality of a connection. In the context of server selection, another important characteristic of a network connection is the bandwidth available to clients of that connection. All other things being equal, higher available bandwidth implies faster document transfer time. Available bandwidth depends on two things: 1) the underlying capacity of the path between client and server which is limited by the slowest (or *bottleneck*) link, and 2) the presence of competing traffic (*congestion*).

These two useful pieces of information are not readily available to applications. In order to discover this information we developed two tools: *bprobe*, which measures the uncongested bandwidth of the bottleneck link of a connection; and *cprobe*, which estimates the current congestion along the bottleneck link of the path.

Armed with measurements of the current network state we can perform application-level congestion avoidance. By application-level congestion avoidance we mean that applications can use estimates of traffic volume to actively avoid paths that are currently heavily loaded. Specifically, in this paper we examine how a combination of measurements of latency, bottleneck link speed, and congestion can be used to formulate a dynamic algorithm for server selection. We show that the combination of all three measurements improves our dynamic server selection policy over any one metric alone.

In the following sections we first motivate and describe our bandwidth and congestion measuring tools. Then we move on to describe experiments that combine these two probe methods with RTT measurements in the context of the server selection problem. We conclude with a discussion of ongoing and future work.

1.1 Related Work

In [9], Gwertzman and Seltzer propose a replication technique (and presumed server assignment) based on geographical distance in which servers are responsible for placing replicas near sources of high demand. By correlating IP addresses and zip codes, the geographical location of hosts can be roughly established, thus allowing the calculation of distance between hosts. This information is used for both replica placement and server selection. In that scheme, a client makes a request to the home server for the address of the nearest replica, and the server calculates distances in miles to find the replica closest to the requesting client. This is essentially a static method of server selection: although new replicas may come on-line from time to time, a request from a given client will usually be forwarded to the same server. This method also relies on the server maintaining a large amount of geographical information. In contrast, we show below that a dynamic server selection policy can provide high performance and allows simplification of the placement policy.

Guyton and Schwartz also focus on distance measures in [8] with the objective of keeping communication local and limiting the use of long-haul links. Since they seek to avoid high-speed links, their approach does not give preference to minimizing round-trip times. They find that the high variability of round-trip times renders them unattractive in practice. However, we show this high variability is an essential feature that can be exploited when selecting a server. Instead of using hops or round-trip time, all of the methods explored in that work attempt to determine a subset of the network topology and build a distance metric on that topology. Hence, the result is again a static policy, and the topology required is gathered at a high cost in some cases. In contrast, our dynamic server selection polices do not require explicit knowledge of topology. Furthermore, as we will show, the quickly changing and highly variable conditions in the Internet require a degree of dynamic assessment that static policies do not provide.

A dynamic server selection policy based on measured latency was used to choose among hierarchically organized caches in Harvest [6]. The cache's resolution protocol performs a remote procedure call to all of the siblings and parents of the cache and checks for hits. The cache then retrieves the object from the responding site with the smallest measured latency. This is similar to the policy we call Dyn 1 below, which uses the elapsed time for a single *ping* to each candidate server and chooses the server which responds first. In this paper, we show that the additional time required for more complete measurement of current network conditions often results in improved performance.

2 Measuring Bandwidth and Congestion at the Application Level

In order to minimize the time between document request and document arrival in a dynamic server selection scheme, the client needs to evaluate the quality of the connection to each of the candidate servers. In particular, a measure of network utilization is necessary to assess potential transfer time. Utilization depends on both the base bandwidth of a path and the presence of other traffic competing for the path.

We use the term *base bandwidth* of a connection to mean the maximum transmission rate that could be achieved by the connection in the absence of any competing traffic. This will be limited by the speed of the bottleneck link, so we also refer to this as the *bottleneck link speed.* However, packets from other connections may share one or more of the links along the connection we want to measure; this competing traffic lowers the bandwidth available to our application. We use the term *available bandwidth* to refer to the estimated transfer rate available to the application at any instant. In other words, the portion of base bandwidth which is not used by competing traffic. We define the *utilization* of a connection as the ratio of available bandwidth to base bandwidth, that is, the percentage of the base bandwidth which should be available to the application.

If a measure of current utilization is available, applications can make informed resource allocation decisions. For the specific problem of WWW server selection, knowledge of current network utilization allows better prediction of information transfer time from the candidate servers.

With this objective in mind, we set out to develop tools to measure the current utilization of a connection. These measurements are necessarily done from a distance and in a unknown environment. Therefore, we refer to our measurement process as *probing* and the tools as *probes*. Our design goals for the probes were that they:

- should work at the application level; *i.e.* by usable by any user program;
- should have a minimal impact on both the network and the server we are probing;
- should not rely on cooperation from the network or the server;
- should be robust and perform properly under varying conditions; and
- should be as accurate as possible while providing timely information.

Our two probe tools are: *bprobe*, which estimates the base bandwidth or bottleneck link speed of a connection; and *cprobe* which estimates the current congestion of a connection. The following sections describe the design and validation of each of them. Further details on both BPROBE and CPROBE can be found in [5].

2.1 BPROBE: Measuring Base Bandwidth

2.1.1 Measurement Theory

In the following discussion we assume a network like the Internet. In such a network, a *path* between any two hosts in the network is made up of one or more *links*. Between each set of links along the path is a *router* which examines the destination address of each packet and forwards it along the appropriate outgoing link. Our main requirement of the network is that packets are not frequently reordered in transit. We also assume that the path is stable, by which we mean that the path packets take at any instant will not change for the next few seconds, at least. For the Internet, routing table updates are infrequent enough that this is a reasonable assumption. We further assume that the bottleneck in both directions is the same link, although this assumption could be relaxed in a different design.

Recall that the goal of BPROBE is a measurement of the base bandwidth of a connection: the speed of the bottleneck link. The essential idea behind the probe, then, is this: if two packets can be caused to travel together such that they are queued as a pair at the bottleneck link, with no packets intervening between them, then the inter-packet spacing will be proportional to the processing time required for the bottleneck router to process the second packet of the pair. This well-known effect is illustrated in the familiar diagram shown in Figure 1

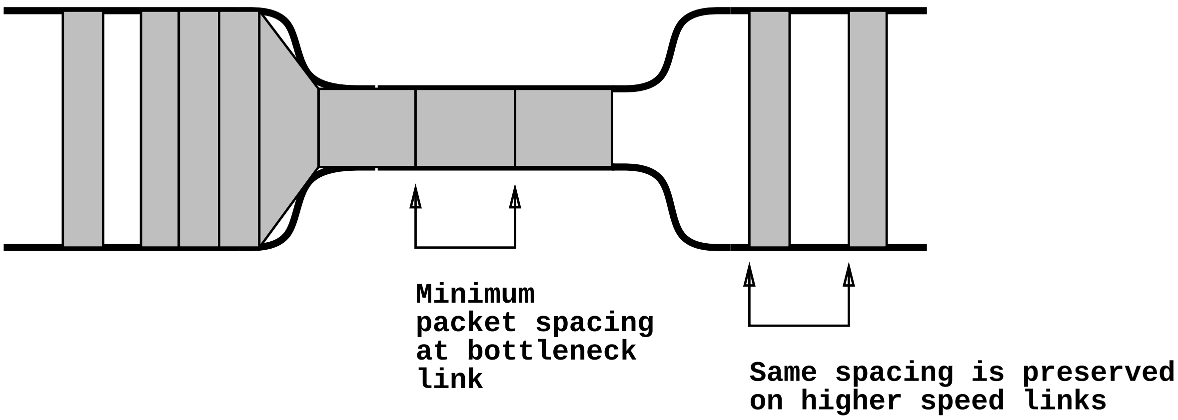

Figure 1: Packet flow through a bottleneck link

(adapted from Van Jacobson [10]). In addition, Bolot describes the basic effect in [3, 4] and Keshav has used a similar method in networks of Rate-Allocating servers [11, 12].

The goal of the BPROBE tool is to create this condition and to use it to make reliable measurements. In other words, if packets from the probe tool alone are queued at the bottleneck link, then the inter-arrival time of those pairs that were queued can be used at the endpoint of the path to estimate the base bandwidth of the bottleneck link. Under ideal conditions, the receiver can use this information to measure the bottleneck link speed as follows: the trailing edge of the first of a pair of packets marks the time when the router started processing the second packet of the pair and the trailing edge of the second packet records the time when the router finished processing that packet. Given a packet of size $\mathcal{P}$, and the inter-arrival time (or *gap*), we can estimate the base bandwidth of the bottleneck link, $\mathcal{B}_{bls}$, as follows

$$\mathcal{B}_{bls}\ \textbf{bytes/second} = \frac{\mathcal{P}\ \textbf{bytes}}{gap\ \textbf{seconds}}.$$

2.1.2 Obstacles to Measuring Base Bandwidth in Practice

However, in contrast to the the network architecture assumed in [11], in our experimental environment (the current Internet) this ideal behavior is not easily achieved. There are several problems that arise in practice:

- **Queuing failure:** Probe packets may not queue at the bottleneck link.
- **Competing traffic:** Competing traffic along the path may intervene between probe packets.
- **Probe packet drops:** Packets sent by the probe may be dropped.
- **Downstream congestion:** Congestion at routers downstream from the bottleneck may invalidate the results.

Each of these problems can be the cause of spurious estimates of base bandwidth. The major obstacle to implementation, then, is this: given a set of inter-arrival time measurements, how can the probe tool decide which will result in valid bandwidth estimates? The challenge was to start from the intuitive idea captured in Figure 1 and design an accurate, robust and low-impact tool to measure bandwidth. We now present our solutions to these problems.

2.1.3 Solutions to Implementation Problems

Both BPROBE and CPROBE are built on top of the ICMP ECHO mechanism. Because of our use of ICMP ECHO, a client can send packets to a host and receive replies without installing new software at the remote site, which affords wide utility of our tools.

There are two principal techniques with which we attack these problems. First, the use of multiple packets of varying sizes; second, the tool uses a careful filtering process which discards inaccurate measurements.

Queuing failure: The first problem to overcome is that the client sending the probe packets may not happen to send data fast enough to cause queuing at the bottleneck router. In order to ensure such queuing, we send a number packets and we use packets of varying sizes. Larger packets will naturally take more processing time at routers and increase the possibility of queuing. Therefore, we want to use as large a packet as can negotiate the round-trip path. The upper limit on packet size will vary from path to path, however, so we use packets of varying size.

Currently, the probe runs in distinct phases with each phase using packets of successively larger sizes. The first phase sends a number of packets (currently 10) of the smallest size that will be used (currently 124 bytes). The next phase uses even larger packets and this process continues until we reach the maximum packet size which our test client can send (approximately. 8000 bytes). In this way, we adapt to the maximum feasible size for each connection.

Competing traffic: The second problem is the interference of other traffic sharing a link along the path. In the ideal case, no non-probe packets intervene. One or more intervening packets will cause an increase in the inter-arrival time and therefore an underestimate of the bottleneck link speed.

The solution to this problem is two-fold. By sending a large number of packets, we increase the likelihood that some pairs will not be disrupted by competing packets. Even when many pairs are disrupted, the number of intervening bytes will often vary from pair to pair. This results in differing bandwidth estimates which may then be filtered to determine the correct estimate. We explain the filtering process below.

As a further measure, we increase the packet size by alternating factors of 150% and 250%. This ensures that no two packet sizes are integer multiples of each other. If we simply doubled the packet size, the bandwidth estimated when one packet of size x intervenes between probe packets of size y would be the same as the bandwidth estimated when two packets of size x intervene between probe packets of size $2y$. As will be clear from the discussion of our filtering method, this could easily result in an erroneous final estimate. The use of alternating increments diminishes the probability of a bad estimate.

Probe packet drops: The third problem that must be addressed is dropped probe packets. Some packets are simply lost, but large packets are more likely to cause buffer overflow and be dropped. Large packets will be fragmented and this also increases the likelihood of dropped packets due to fragment loss. (Aside from this effect, fragmentation of large packets has not been a problem for our tools.) We avoid this problem by sending packets of varying sizes. Sending many packets of each size also serves to make the probe tolerant of a few packet losses regardless of the cause.

Downstream congestion: The fourth problem occurs when queuing occurs on the return trip, after passing through the bottleneck link. That is, even if all goes well from client to the server and back through the bottleneck link, congestion at intermediate servers downstream from the bottleneck can invalidate an estimate. Consider a pair of packets whose inter-packet gap was properly set by the bottleneck link. If these packets subsequently get queued, the inter-packet gap will be reset, thus measuring the wrong link. Since this subsequent queuing is unlikely, we can alleviate this problem during filtering. Some of the pairs may, in fact, measure the wrong link but if enough of the pairs make it back without further queuing, the erroneous estimates will be filtered out.

The Filtering Process: The most difficult problem that must be addressed by the probe is identification of those estimates that should be used to determine the base bandwidth from those that should be discarded due to lack of queuing, subsequent queuing or interference from competing traffic. Each phase of 10 probe packets results in at most 9 inter-arrival measurements. Thus, at most 7 sets of at most 9 measurements each are the input to the filtering process. Given the packet size (and some intelligent guesses as to protocol headers and link level headers and trailers) a direct estimate can be made of the bandwidth *assuming queuing* as defined before.

For example, Figure 2 shows the raw estimates for 5 invocations of the probe tool. All of the data points belonging to one invocation are shown by plotting a numeral representing the invocation number at each data point. The abscissa of a data point is the bandwidth estimate, and the ordinate increases with packet size. Thus, all the measurements for the first invocation occur on the bottom 5 line segments of the graph, all those for the second on the next higher 5 lines and so on. Within each invocation, packet size is increasing from bottom to top. Notice the clustering of estimate values as packet size increases; this shows that, in general, larger packet sizes yield more consistent results.

The multi-phase variable-packet-size design of the probe results in 1) correlation among correct estimates and 2) lack of correlation among incorrect estimates. For example, suppose all intervening packets are of size x, and the probe packet sizes for two probe phases are $p1$ and $p2$ with $p1 \neq p2$. The estimates produced by the packet sequences $p1 - x - p1$ and $p2 - x - p2$ will both underestimate the base bandwidth, but the important characteristic is that the two estimates will *differ*. However, sequences like $p1 - p1$ and $p2 - p2$ will both produce agreeing correct estimates. Other incorrect estimates such as produced by the sequence $p1 - x - x - x - p1$ will be even less correlated with the correct estimates and also uncorrelated with other. It is these properties of the data, which are evident in Figure 2, that we seek to exploit using non-linear filtering.

We have tested two filtering methods both of which rely on computing an "error interval" around each

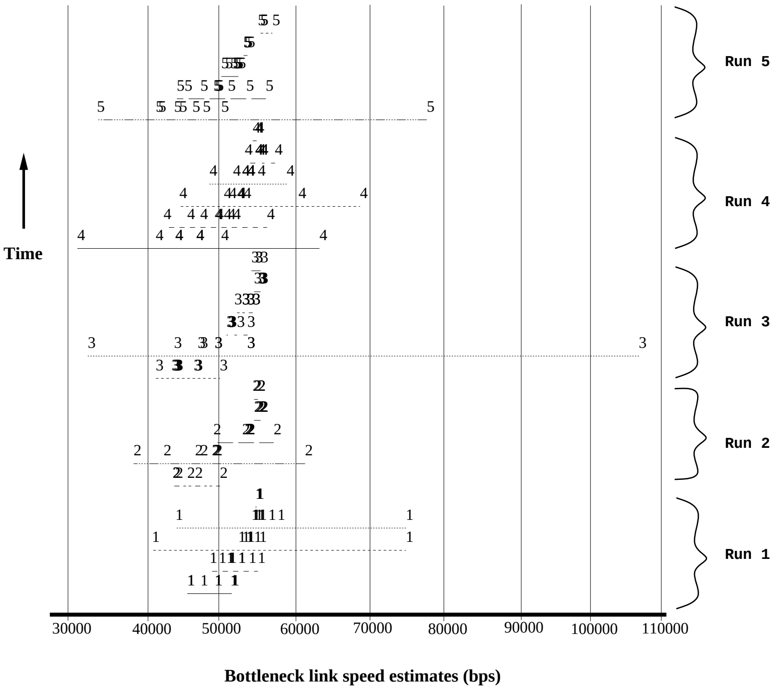

Figure 2: 5 BPROBE experiments to local 56kb hosts

estimate, and subjecting these intervals to further set operations as explained below. The error interval can be expanded as necessary until a satisfactory estimate of the speed of the bottleneck link is determined. The *intersection* filtering method finds overlaps between estimate intervals and computes the intersection, with the idea being that we want to find the estimate that occurs in all sets. Since we observed that larger packets provide better estimates we start intersecting using the estimates from the largest packets and iteratively intersect with estimates derived from successively smaller packets. The *union* filtering method combines overlapping intervals using set union, and selects an interval only if enough sets contribute to it. Both methods produce one interval as the final result and the midpoint of this interval is returned as the final estimate. In our experience the union approach shows less dispersion of estimates than intersection and we have adopted it as the filtering method currently used by BPROBE

2.1.4 Validation of BPROBE

To validate the base bandwidth probe tool, we performed three sets of experiments. We tested BPROBE between a fixed host and hosts on our local network, on our regional network, and on the wide-area network. For each of the networks we used available maps of link capacities to determine the bottleneck link speed for each of the target servers. Therefore, we were able to compare the results of the tool against the known bottleneck link speeds.

First, we tested the probe to a set of 16 hosts on our local network, 9 of which were connected by Ethernet and 7 of which were connected over 56Kbps lines. Each minute over a period of 4 days we ran the probe tool and recorded the bandwidth estimate. The results of these tests are summarized in Figure 3, which presents histograms of BPROBE bandwidth estimates for the two classes. The histogram bins are labeled with bottleneck link speeds in bits-per-second. On the left of the figure is the histogram of the estimated bottleneck link speeds for the 7 56Kbps hosts; on the right, is the histogram of the estimated bottleneck link speeds for the 9 Ethernet (10Mbps) hosts. The figure shows the majority of the estimates closely clustered about the expected values, even though the expected values differ by two orders of magnitude.

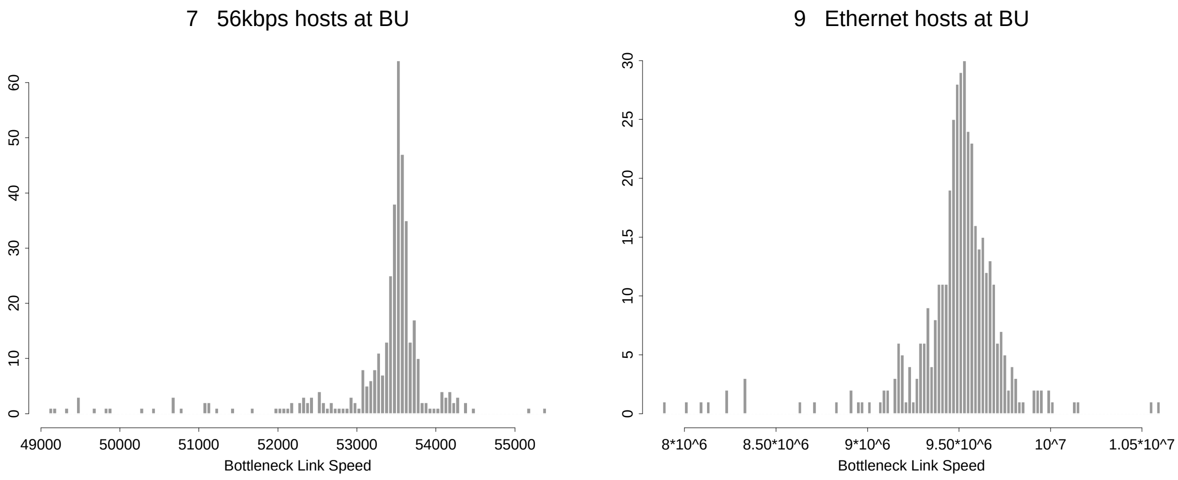

Figure 3: Histograms of BPROBE results. Left: 56Kbps; Right: 10Mbps

Relative error values for the local hosts along with results for regional and wide-area hosts are presented in Table 1, which presents a more quantitative view of this data. The columns show percentages of error relative to the rated capacity of the bottleneck link; each row shows a set of hosts. Each entry in the table gives the percentage of the bandwidth estimates that fall within a given relative percentage of the rated capacity. For example, 96% of the bandwidth estimates of local 56Kbps hosts were within ±10% of 56Kbps. As is evident from Figure 3 there is a systematic underestimation error due to unaccounted for link-level encapsulation overhead. In practice, this should not be a significant problem. For example, this could be addressed by using a list of commonly available capacities to round the bandwidth estimates to the nearest likely value.

The results of these three sets of experiments convince us of the accuracy of the probe tool. For three very different classes of hosts we were able to accurately measure the base bandwidth of links using BPROBE. As we move from local to regional and then wide-area hosts, the tool shows only a minor loss in estimate accuracy.

Considering now our stated goals, we find that BPROBE does provide accurate base bandwidth estimates in spite of the difficult environment it measures; it operates without explicit cooperation from the network or servers; and it provides reliable estimates without excessive time overhead. At the current time, a reasonable concern is the probe's impact on the network. The current implementation consumes too much network bandwidth. Two approaches are under development: First, we believe we can achieve nearly the same accuracy and reliability

			Relative Error		
Network	Rated Capacity	No. Hosts	±5%	±10%	±20%
Local	56Kbps	7	55%	96%	99%
	10Mbps	9	80%	97%	100%
Regional	56Kbps	6	73%	87%	92%
	1.54Mbps	9	23%	52%	75%
	10Mbps	3	68%	86%	92%
Wide-area	56Kbps	4	84%	93%	97%
	1.54Mbps	3	12%	54%	82%
	10Mbps	3	31%	53%	63%

Table 1: Measurements of BPROBE accuracy for local, regional and wide-area host sets.

with fewer packets; and second, it is anticipated that the base bandwidth estimate of a path will be cached, so that redundant measurements of a path will be unnecessary.

2.2 CPROBE: Measuring Available Bandwidth

In the previous section we described our bandwidth probing tool which gives a fairly reliable estimate of the bottleneck link speed for a path between two hosts on the Internet. We now consider estimating available bandwidth.

To measure available bandwidth, we developed a tool called *cprobe*. CPROBE's technique is straightforward: by bouncing a short stream of echo packets off of the target server and recording the time between the receipt of the first packet and the receipt of the last packet, we can measure the presence of competing traffic on the bottleneck link. Dividing the number of bytes sent by this time yields a measure of available bandwidth, $\mathcal{B}_{avail}$, that represents the actual throughput achieved by the probe bytes. As long as we can send packets at a higher rate than the bottleneck link speed (which we can measure using BPROBE) this effect should occur. Any additional time lag between the first packet and the last one represents non-probe bytes (competing traffic). In order to tolerate packet drops and possible re-ordering of packets, we use the results of four separate 10-packet streams when calculating the available bandwidth.

The utilization of the bottleneck link can then be computed as the ratio of available bandwidth measured by CPROBE to the bottleneck link speed as estimated by BPROBE:

$$\mathcal{U}_{probe} = \frac{\mathcal{B}_{avail}}{\mathcal{B}_{bls}}.$$

As in the case of BPROBE, care must be taken to ensure that the CPROBE's results are valid. We validated the individual inter-arrival measurements using a packet tracing tool running on a local Ethernet. The experimental set-up consisted of the probe client and the probe target; a host running the packet trace tool; and other hosts between which FTP sessions were run to provide a background load. While varying the background load we ran several repetitions of the probe tool. We then compared the probe measurements with the log of the packet traces. The probe's measurements of elapsed time for packet transfers were accurate to within $\pm 10\%$ relative to the packet trace. However, there were occasional irregularities in the data. For example, in some cases, the flow of returning packets would be delayed, if the sending host were momentarily interrupted. To eliminate these readings from our data we discard the highest and lowest inter-arrival measurements when calculating $\mathcal{B}_{avail}$.

3 Dynamic Server Selection

3.1 Why Dynamic Server Selection?

In this section, we demonstrate how our tools can be used in the context of a specific instance of the server selection problem: minimizing the response time for replicated WWW documents. In a dynamic server selection scheme, a client seeking a document should choose the server which it believes will deliver the document in the shortest amount of time. In this section, we show that the impact of this choice is strongly affected by whether it is made based on static assignment or based on dynamic measurements of current conditions.

The difference between dynamic and static approaches to server selection is illustrated by the measurement of distance. Two obvious metrics for measuring distance in the Internet are hops, and round-trip latency. However, the static nature of the hops metric (the number of hops between any two hosts rarely changes) is in considerable contrast to the quickly changing, widely varying metric of round-trip latency. These differences are of great importance in the server selection problem.

We began by comparing empirically measured distributions of the two. The data sets were generated by measuring the values of both metrics between a fixed client host and 5262 servers selected randomly from a list of WWW servers [13]. Figure 4 shows on the left, the measured distribution of hops to the set of servers; and on

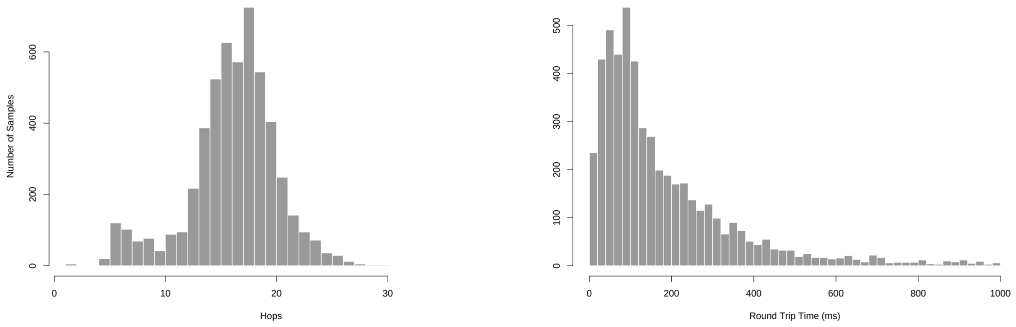

Figure 4: Empirical Distribution of Hops and RTT to 5262 Random Servers

the right, the distribution of round-trip latencies (in ms, measured using *ping*). These distributions are strikingly different. The distribution of hops appears to be fairly symmetric about its mean, whereas the distribution of latencies has a median (125) much less than the mean (241). The differences between the distributions suggest that hops is a poor predictor of latency and in practice we find that this is the case.

The fact that hops is not a good predictor of latency suggests that either variation in link speed or delays due to congestion dominate the round trip time of packets. The impact of congestion is made clear by examining time series plots of round trip times to a single host. Unsurprisingly, these time series plots show extreme variance over short time periods. Consider Figure 5, which presents measurements of latency to a single host gathered over a period of two days at intervals of 30 seconds. On the left is a time-series plot; on the right a histogram of the same data. The variation in latency measurements reflects the underlying changes in congestion. It is these effects which can be exploited by a dynamic server selection policy.

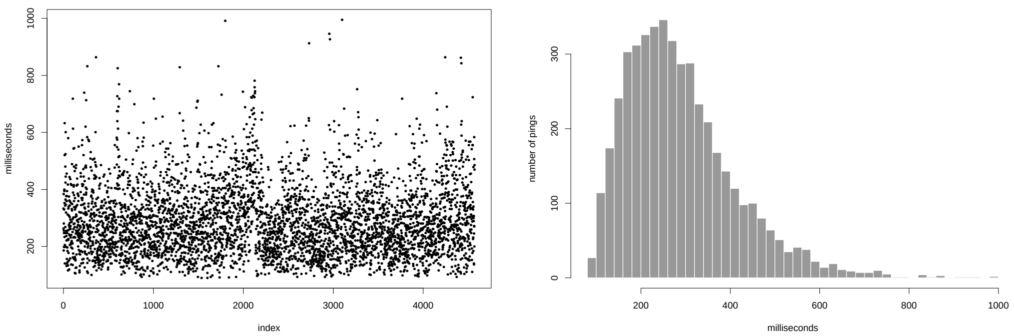

Figure 5: Round Trip Times to a Single Host

The difference in the behavior of dynamic and static server selection policies can be explained by considering the time-varying nature of these two distance measures. However, the difference between the characteristics of hops and latencies is fundamental enough to also suggest differences in algorithms for server replication. An initial replica placement policy might try to place replicas "close" to the clients that will use them (*e.g.,* [9]). This follows naturally when thinking about hops as the distance metric. Because the bulk of the hops distribution is centered about the mean, care is required in placing replicas if the goal is to minimize the number of hops between client and server. In other words, a random distribution of replicas will not significantly change the mean distance in hops between clients and the servers they use. On the other hand, this is clearly not the case when the measure is round-trip latency. Because the bulk of the probability mass of the latency distribution lies below the mean, a random placement strategy should markedly reduce the mean latency between client and server.

Taken together these observations indicate that the performance of replicated server system that uses dynamic server selection based on round-trip latency will be less sensitive to replica placement than a system that relies on the static approach where hops is the measure of distance.

As a demonstration of this idea, suppose that m documents are each to be replicated on a different set of n servers. Also, assume that each document has a home server that maintains a list of the locations of each replica. When a client requests a service it retrieves the list of alternate sites from the home server and applies a dynamic server selection policy to choose the nearest from among the alternatives. We want to know what the average distance will be, both in hops and in round-trip latency, measured over all m documents. In order to assess the difference we performed to following simple experiment using our dataset of hops and latency measurements, fixing $m = 1000$ and varying n. For a given replication level n, we selected n random samples from our empirical distributions of hops and round-trip latency. For each trial the result is the minimum of the n values. The average of m trials is the final result.

The results are shown in Figure 6. The graph on the right shows results obtained when the number of replicas is varied between 1 and 200; on the left we zoom in on values up to 20. The y axis give the distance between client and server as a fraction of the mean value. This figure shows that the distance as measured statically using hops decreases slowly and never drops much below 25% of the mean. In contrast, the distance as measured dynamically using round-trip latency between client and dynamically chosen server drops sharply, approaching less than 10% of the mean even for small degrees of replication, indicating that a random replica placement policy coupled with dynamic server selection could be very effective at reducing round-trip latency.

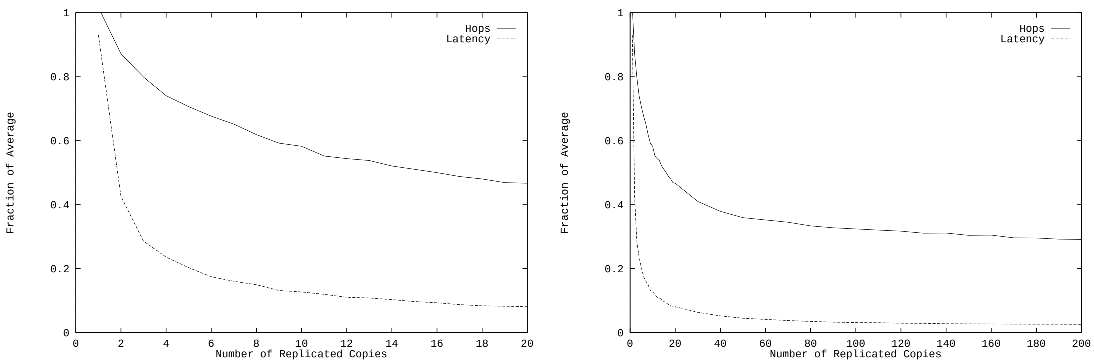

Figure 6: Performance of Random Replica Placement with Dynamic Server Selection

3.2 Simple Approaches to Server Selection

In this section we present experimental results to show that even very simple dynamic server selection policies can outperform static approaches in the real Internet. As previously stated, our dynamic policy operates under the assumption of widespread replication of documents (such as proposed in [1, 7]). We further assume that it is possible for a client to obtain a list of hosts serving replicas of the document of interest (perhaps by contacting the home server of a document).

In order to simulate such an system, we identified 10 hosts with approximately equal average round-trip latency as measured from our test client. Then, over a 3-day period, we periodically measured the round-trip latency using *ping*, and measured the transfer time from each host for documents of sizes 1KB, 5KB, 10KB and 20KB. Using this data we then simulated several server selection policies: 1) Static, based on minimizing geographic distance; 2) Static, based on minimizing the number of network hops between client and server; 3) Dynamic, based on random selection of a server; and 4) Dynamic, based on minimizing the the mean of 1,2,3,4 or 5 round-trip measurements. The results are summarized in Figure 7.

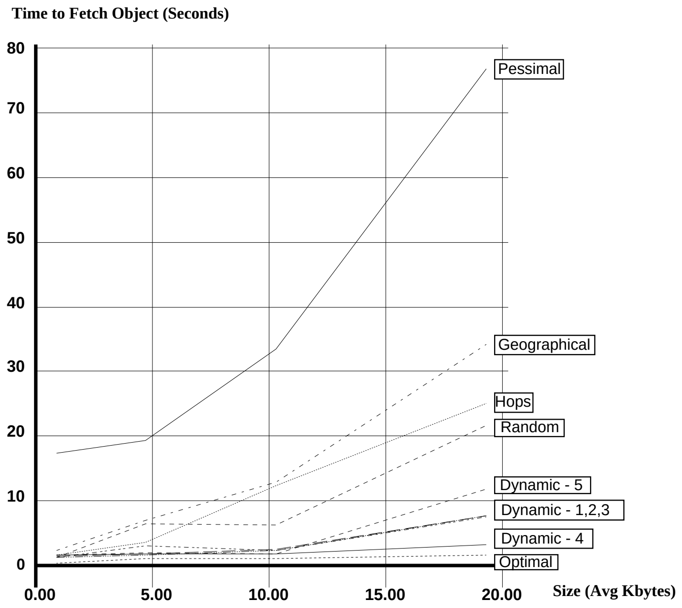

Figure 7: Fetch Times of Static and Dynamic Policies Plotted Against Document Sizes.

All of these policies were evaluated in terms of user response time with the objective of minimizing the time between document request and document arrival. In this respect, as can be seen in Figure 7, the dynamic policies consistently outperformed the static policies and the benefit of a dynamic policy generally increased with document size. Even random selection is preferable to static polices for documents larger than 5KB. The policy based on the mean of 4 round-trip time measurements, the best dynamic policy in this simulation, was only 4 times the optimal transfer time for 5Kbyte files and this improved to less than 2 times optimal for larger files. In contrast, the static hops policy which was also about 4 times optimal for small files, showed increasingly worse performance as transfer size increased, up to 20 times optimal for 20KB files. These results use 7 replicas, certainly not a very large number for WWW document replication. The results would surely improve with a greater degree of replication as suggested earlier by Figure 6.

However, the benefit of multiple measurements of round-trip time decreases with increasing file size. We believe that this indicates the effect of limited bandwidth due to competing traffic, which will have a greater impact on larger document transfers, while small document transfers would have their transfer time dominated by instantaneous latency. This observation led us to formulate the techniques described earlier for the measuring base bandwidth and available bandwidth of a path. We next discuss dynamic server selection using these measures in addition to latency.

3.3 More Detailed Approaches to Server Selection

The results presented in the previous section clearly show the benefit of dynamic policies. However, the relative benefit was observed to decrease with increasing transfer size. Since we suspected that one of the factors involved in this fall-off in predictive ability was bandwidth limitation, we developed the tools presented in the first part of the paper to enable estimation of both uncongested link capacity and instantaneous congestion. In the context of server selection for WWW documents, given these measurements and the size of the document to be retrieved, it is possible to estimate the transfer time directly. If the document size is not known, then a choice could be made based on, for example, highest available bandwidth. Given the bandwidth probing tools in addition to the RTT estimators commonly available, the question becomes: can we improve the predictive capability of our dynamic server selection algorithm?

In order to evaluate server selection policies based on available bandwidth measurements, we collected data from several WWW servers in the Internet. From earlier work [2] we have available a database of document size and location information. From this source we extracted sets of documents of sizes 100KB, 500KB, 750KB and 1MB. We chose these larger sizes because the transfer times of these large documents should be influenced by available bandwidth as well as latency. Periodically, over a several hour period we recorded the following data for each document: 5 round-trip time measurements (using *ping*), the bottleneck link speed (using BPROBE), the link utilization (using CPROBE) and the transfer time for the document.

Before beginning simulation of the various selection policies, we performed regression analysis on various combinations of factors for each of the four data sets. In particular, for each document size, we used linear regression to analyze the dependence of measured transfer time on: 1) the RTT as measured by *ping*; and 2) the predicted transfer time using a combination of RTT and the bandwidth probing tools.

	Document Sizes			
Regression Formula	100K	500K	750K	1MB
Measured transfer time against single ping	0.0532	0.1866	0.4778	0.0812
Measured transfer time against predicted TT	0.2912	0.2807	0.6386	0.1451

Table 2: Regression Analysis: R^2 values for two metrics for 4 document sizes

Table 2 gives the R^2 values for the regression calculations. The regression analysis shows an improvement in accuracy of predicted transfer time when the measurements from the bandwidth probing tools are added to latency measurement provided by a single ping. Also the predictive value of our measure increases with larger document size, except for the very largest documents. We suspect that the long transfer times associated with the largest documents extend beyond the valid prediction window. As we have shown above, the effect of competing

	Transfer Sizes			
Selection Policy	100K	500K	750K	1MB
Pessimal	21.836	21.800	18.357	696.680
Random	6.857	7.989	10.572	281.919
Hops	2.267	8.958	10.120	46.190
Predicted TT	2.712	2.001	5.721	31.897
Optimal	0.985	1.296	4.265	17.529

Table 3: Simulation results for large transfers - mean transfer time in seconds

traffic on available bandwidth is a highly variable one. Therefore, estimates have a limited useful lifetime; we are currently exploring methods for assessing those limits.

Our new policy, called "Predicted TT" (for Predicted Transfer Time), computes the predicted transfer time to all servers and chooses the minimum value. We use the regression coefficients k_1 and k_2 which relate the predicted transfer time to round-trip latency and a bandwidth-limited component, as follows:

$$\text{PredictedTransferTime} = k_1\text{RTT} + k_2\frac{\text{document size}}{\mathcal{B}_{bls} * \mathcal{U}_{probe}}$$

In the simulation of this selection policy, the predicted transfer time is calculated for all candidate servers and the server with the minimum value is chosen

Using the outputs of BPROBE and CPROBE we can calculate a value for the predicted transfer time of a document as follows: First, measure the round-trip latency; Second, compute the available bandwidth as the product of the estimates of base bandwidth and utilization: $\mathcal{B}_{avail} = \mathcal{B}_{bls} * \mathcal{U}_{probe}$; Third, compute the bandwidth-limited component of the transfer time by dividing the document size by the available bandwidth: $BandwidthComponent = documentsize \div \mathcal{B}_{avail}$; and finally, use the regression coefficients to weight the two factors of RTT and bandwidth component: $Value = k_1 MeanPingTime + k_2 BandwidthComponent$.

3.3.1 Evaluating Detailed Server Selection

We again simulated both static and dynamic server selection algorithms using the data collected from our survey of WWW servers. For each document size (100K, 500K, 750K and 1MB) we selected 7 data points uniformly from the recorded data and applied the dynamic algorithm based on round-trip measurement and available bandwidth to make a selection from among the 7 sites. The results are presented in Table 3. Each column gives the results of applying each server selection policy for one particular document size. Each row gives the results for a particular server selection policy for all 4 document sizes. Each entry in the table represents the mean transfer time in seconds over 1000 simulated server selection decisions: that is, transfers of documents of the size given by the column heading when the server is selected according to a policy given by the row heading.

The policies simulated were: 1) Pessimal, which simply chooses the worst (largest) transfer time among the 7 servers; 2) Random, which picks one of the 7 servers with uniform probability; 3) Hops, which chooses the server which is the fewest hops away; 4) Predicted TT, as defined above; and 5) Optimal, which chooses the best (smallest) transfer time.

Because we expect that, in practice, the base bandwidth of a connection will be cached, we do not include the cost of running BPROBE in the simulated transfer time. We do however, include the time required for measurement of round trip latency in all polices that use this measure.

The superiority of the dynamic policy for server selection is clear from Table 3. The improvement is especially marked for large documents. For example, for 500KB documents, an improvement of over 75% is found when using Predicted TT rather than Hops (2 seconds versus nearly 9 seconds). The transfer time chosen by the dynamic policy is less than 2 times the Optimal time for large documents, while the Hops policy results in transfers taking at least $2\frac{1}{2}$ times the Optimal time.

3.3.2 RTT vs Predicted TT

Previously we showed that the bandwidth measurements combined with latency showed greater predictive capability than latency alone. We now assess the effect on the actual performance of dynamic server selection.

In Table 4 we add two more dynamic policies: 1) Dyn 1, which performs a *ping* to each server and chooses the server with minimum RTT; and 2) Dyn 5, which computes the mean of 5 *ping* measurements and chooses the server with the minimum value. Both of these policies were also used in the initial study described in section 3.2. Once again, this table shows that a dynamic policy is always superior to the static policy based on distance measured using hops, confirming our earlier results. Also, for large files, the use of the probe measurements improves over the results from a single latency measurement.

The surprising result is that the Dyn 5 policy which relies solely on round-trip latency measurements gives results very close to those of Predicted TT. One conclusion that can be drawn from this data set is that the policy of minimizing the mean of 5 *ping* measurements already accounts in a way for congestion effects. It appears that direct measurement of congestion as we have done with CPROBE, can by simulated to some degree by using a sequence of RTT measurements. This is significant because of the (currently) high overhead of the probe measurements. However, although the Dyn 5 policy shows good results in the experiments, since it is not founded on clear principles in the way that CPROBE and BPROBE are, it is unclear whether Dyn 5 will perform as well under more general conditions.

3.3.3 Impact of Server Load

As a validation step in our data analysis we studied the correlation between measured transfer rate and available bandwidth. Ideally, of course, plotting measured transfer rate against available bandwidth would yield the line $x = y$. In practice, we found an overestimate by a factor of about 2. We believe that this overestimate of real transfer rate can be explained in large part by the overhead of the TCP protocol. Correcting for this factor

	Transfer Sizes			
Selection Policy	100K	500K	750K	1MB
Hops	2.267	8.958	10.120	46.190
Dyn 1	2.124	2.147	6.612	50.857
Dyn 5	2.301	2.028	5.762	31.175
Predicted TT	2.712	2.001	5.721	31.897

Table 4: Simulation results for large transfers - mean transfer time in seconds

we then found a significant set of hosts along the predicted $x = y$ line. However, there was also a large set of hosts for which available bandwidth was still seriously overestimated by the probes. Looking more closely we identified a number of sites which were known to be popular such as *wuarchive.wustl.edu, www.ncsa.uiuc.edu* and *sunsite.unc.edu*. This suggests that server load should be an input to a dynamic server selection algorithm.

As a first attempt to measure server load we tried the following technique: we fetched a non-existent document from the target WWW server and measure the time required. In general, this experiment results in retrieving a small (less than 200 byte) "error:URL not found" document. In a few cases, larger, personalized replies were received. There is some correlation between the larger fetch times and serious overestimations of transfer rate, confirming our hypothesis that server load is an important factor in transfer time. However, our simulation results do not show improvement for this policy due to the overhead of this method of server load measurement. In fact, our simulations show no improvement over the Predicted TT policy. In practice, the cost of fetching a test document outweighs any benefit in improved prediction of transfer rate for the actual document.

A second simulation run on those data points lying close to the predicted line shows much better results, unsurprisingly. In fact, with this reduced data set of 100KB documents, the Predicted TT policy (0.616 seconds) improves on both the Dyn 1 (0.67 seconds) and Dyn 5 (0.89 seconds) polices. This result underscores the need for an inexpensive means of identifying the heavily loaded servers. What is needed is a lightweight server load measurement method which can quickly and accurately assess server load. Such a tool could be integrated with the tools we present here to significantly improve the server selection process.

4 Future Directions and Conclusions

We have plans to extend this work in several directions. Currently, we are building a version of the Mosaic WWW browser which uses the BPROBE and CPROBE tools to inform the user of the relative expected transfer speeds of links on a Web page. The hypertext anchor is color-coded with a measure of the current network state. For instance, faster expected transfers are coded green while slower ones are red.

Another browser extension we are considering is support for multiple URLs per link in a document. This list of alternate servers can be used as input to dynamic server selection by the browser. This represents a step along the way to integrated support for replication in which the browser would have to *obtain* a list of replicas. Such a mechanism might be particularly attractive to heavily loaded sites.

We also plan to package the probe tools as a daemon that can be placed strategically throughout a network to allow probing of conditions in remote parts of the network. This would allow measurement of a link as a part of several paths and could be used to provide confirmation of probe estimates.

In conclusion, we have introduced and validated two tools useful for measuring network conditions: BPROBE which uses ECHO packets to measure the bottleneck link speed of paths between hosts in the Internet; and CPROBE which estimates the congestion along a path. We have motivated the need for dynamic selection of servers by observing that the combination of random placement of replicas with dynamic server selection can result in a reduced average response time for service clients when compared with static policies. We presented simulation results that show the distinct benefit of dynamic server selection over standard static server assignment policies. We then illustrated how BPROBE and CPROBE could be used by clients of replicated services as input to a server selection decision in the context of WWW documents, thus demonstrating the use of our tools in support of application-level congestion avoidance.

References

[1] Azer Bestavros. Demand-based resource allocation to reduce traffic and balance load in distributed information systems. In *Proceedings of SPDP'95: The 7^{th} IEEE Symposium on Parallel and Distributed Processing*, San Anotonio, Texas, October 1995.

[2] Azer Bestavros, Robert L. Carter, Mark E. Crovella, Carlos R. Cunha, Abdelsalam Heddaya, and Sulaiman A. Mirdad. Application-Level Document Caching in the Internet. Technical Report BU-CS-95-002, Boston University, Boston, MA 02215, February 1995. Also in Proc. SNDE'95 (2nd. International Workshop on Services in Distributed and Networked Environments).

[3] Jean-Chrysostome Bolot. Characterizing End-to-End packet delay and loss in the Internet. *Journal of High Speed Networks*, 2(3):305–323, 1993.

[4] Jean-Chrysostome Bolot. End-to-End Packet Delay and Loss Behavior in the Internet. In *Proceedings of SIGCOMM 1993*, pages 289–298. ACM SIGCOMM, August 1993.

[5] Robert L. Carter and Mark E. Crovella. Measuring Bottleneck Link Speed in Packet-Switched Networks. Technical Report BU-CS-96-006, Boston University, 1996.

[6] Anawat Chankhunthod, Peter B. Danzig, Chuck Neerdales, Michael F. Schwartz, and Kurt J. Worrell. A hierarchical internet object cache. Technical Report CU-CS-766-95, University of Colorado - Boulder, Mar 1995.

[7] Yasuhiro Endo, James Gwertzman, Margo Seltzer, Christopher Small, Keith A. Smith, and Diane Tang. VINO: The 1994 fall harvest. Technical Report TR-34-94, Harvard University Department of Computer Science, 1994.

[8] James D. Guyton and Michael F. Schwartz. Locating nearby copies of replicated internet servers. In *Proceedings of SIGCOMM '95*, August 1995.

[9] James Gwertzman and Margo Seltzer. The case for geographical push-caching. In *HotOS '94*, 1994.

[10] Van Jacobson. Congestion Avoidance and Control. In *Proceedings SIGCOMM '88 Symposium on Communitations Architectures and Protocols*, pages 314–329, Stanford, CA, August 1988.

[11] Srinivasan Keshav. A Control-Theoretic Approach to Flow Control. In *Proceedings of SIGCOMM 1991*. ACM SIGCOMM, 1991.

[12] Srinivasan Keshav. Packet-Pair Flow Control. Available at http://netlib.att.com/netlib/att/cs/doc/94/2-17.ps., 1995.

[13] net.Genesis Corporation. Comprehensive list of sites. Available at `http://www.netgen.com/cgi/comprehensive`., April 1995.