An Empirical Evaluation of Client-side Server Selection Algorithms

Sandra G. Dykes, Kay A. Robbins
Division of Computer Science
University of Texas at San Antonio
San Antonio, TX 78249-0667
{sdykes, krobbins}@cs.utsa.edu

Clinton L. Jeffery
Department of Computer Science
University of Nevada at Las Vegas
Las Vegas, NV 89120
jeffery@cs.unlv.edu

Abstract—**Efficient server selection algorithms reduce retrieval time for objects replicated on different servers and are an important component of Internet cache architectures. This paper empirically evaluates six client-side server selection algorithms. The study compares two statistical algorithms, one using median bandwidth and the other median latency, a dynamic probe algorithm, two hybrid algorithms, and random selection. The server pool includes a topologically dispersed set of United States state government web servers. Experiments were run on three clients in different cities and on different regional networks. The study examines the effects of time-of-day, client resources, and server proximity. Differences in performance highlight the degree of algorithm adaptability and the effect that network upgrades can have on statistical estimators. Dynamic network probing performs as well or better than the statistical bandwidth algorithm and the two probe-bandwidth hybrid algorithms. The statistical latency algorithm is clearly worse, but does outperform random selection.**

Keywords—**replication, caching, server selection, site selection, Web**

I. Introduction

Distributed systems rely on replicated objects and services to improve performance and reliability. For example, popular World Wide Web sites are often replicated at multiple locations, and systems are being designed to share cached objects across the Internet [1][2][3][4]. When objects are replicated or cached on multiple servers, the client would like to select a server that offers fast response time. However, fluctuations in network congestion and server load make it difficult to predict response times, especially during busy times of day when network load is the heaviest. The purpose of this study is to compare client-based methods for server selection on the Internet and to explore factors that influence their effectiveness.

Server selection algorithms and their corresponding performance estimates are applicable to areas other than Web caching and replication. Multimedia content providers could offer clients a choice of content fidelity for different estimates of transfer rates. As an illustration, a client-side monitor could use performance estimators to negotiate a video fidelity such as image resolution in order to maintain a fixed frame rate for real-time display. In mobile networking applications, changes in response time for a client-server connection depend upon current client location as well as server load and network cross traffic. A mobile client could benefit from a dynamic method of selecting servers as its location changes.

Selection criteria, or performance estimators, fall into three classes: *static, statistical, and dynamic*. Static estimators are based upon hardware resources and configuration, such as number of hops, connection bandwidths, and server hardware. These estimators take into account resource capacity, but not availability or contention. Response time, however, depends upon both resource capacities and loads.

Statistical estimators are computed from past performance data such as latencies and bandwidths, therefore they reflect typical levels of contention for a server and the network connection. Statistical estimators are less reliable when the data exhibit high variability, as observed for Internet traffic and HTTP server loads [5][6]. When variability is high, more measurements are required to determine a reliable estimator of expected value. Even with a reliable estimator, the high variability produces large errors in performance predictions. For the Web, this problem is compounded by the fact that latencies depend upon time-of-day, and bandwidth depends upon both time-of-day and file size [7]. Still, previous work suggests median latencies or bandwidths might be a viable method for server selection [8][9]. Our experiments include two statistical algorithms, one based upon median latency and the other upon median bandwidth.

Dynamic or run-time estimators use small probes to detect current network and/or server conditions. Probes provide an estimate of current resource availability, but cannot include all performance factors. For example, a `ping` probe measures network latency but does not measure server delay or the effect of dropped packets on TCP/IP mechanisms. Another problem arises if conditions fluctuate more rapidly than the time for the document transfer. In this case, the conditions measured by a dynamic probe will not extend over the lifetime of the transfer. Finally, probes can add run-time overhead, both in terms of traffic and elapsed time. Our experiment includes a dynamic probe algorithm that reflects network and TCP/IP stack conditions, but not HTTP server load.

Algorithms for site selection may rely on a combination of estimators. Two algorithms in the study are hybrids that combine a statistical bandwidth estimator with a dynamic network probe. We anchor the study with a sixth algorithm that uses random selection to pick the Web site.

II. Related work

Server selection methods fall into four categories: router, DNS, server-side and client-side methods (see Table I). This paper focuses on client-side methods in which the clients or their proxies select the servers. Client-side selection is appropriate when the group of servers is heterogeneous or widely dispersed across the network. Conversely, server-side selection meth-

This work was supported by National Science Foundation grant CDA-9633299 and by a fellowship from the NASA/Texas Space Grant Consortium.

TABLE I

CLASSIFICATION AND EXAMPLES OF SERVER SELECTION METHODS. RTT IS NETWORK ROUND TRIP TIME, BW AND LATENCY ARE STATISTICAL ESTIMATORS DERIVED FROM HISTORIC DATA

Category	*Selection metrics*	*Example*	*Ref.*
Client-side	geography, hops	Push-caching	[15]
	RTT	ICP, NLANR	[2][16]
	BW	SPAND	[17]
	random, server load	Smart Client	[18]
	prior response time	empirical study	[19]
	hops, RTT, BW	simulation study	[20][21]
	hops, RTT, latency	empirical study	[9]
	RTT, BW, latency	empirical study	*this work*
Server-side	server load	HTTP Redirect	[8]
		IP address rewrite	[14]
DNS	round-robin	RR-DNS	[12][13]
Routers	router metric	IPv6 Anycast	[10][11]

ods focus on server clusters. Server clusters typically contain members with similar resources and a shared local network. In clusters, the primary concern is balancing request load across servers: when resources are equal and the load is balanced, a client receives similar response times from all servers. One popular approach for load distribution uses DNS aliasing [12][13]. A site is assigned multiple IP records in the local DNS table. Upon receiving a translation request, the DNS Bind program selects from among these records in a round robin fashion (DNS-RR). While simple, DNS-RR offers only crude load balancing and is often combined with a server-side mechanism. In server-side mechanisms, clients send requests to a dispatching module that tracks current load conditions and assigns servers accordingly. The dispatcher may direct a client to the chosen server via an HTTP Redirect [8]. Alternatively, the dispatcher may change IP addresses on all incoming packets to route them to the appropriate servers [14]. This algorithm requires the dispatcher keep track of all the site's TCP streams, earning it the name TCP router.

The last category of server selection methods relies on network routers. IPv6 allows an Anycast address to be assigned to one or more network interfaces. Routers choose among the interfaces by deciding which is "nearest", where nearest is defined by a router metric such as hop count. The problem with Anycast lies in determining a router metric that effectively predicts user response time: several studies have shown that, in general, hop count shows little correlation with bandwidth or response time [7][9][21]. Yoshikawa, et al., argue that "in many cases the client, rather than the server, is the right place to implement transparent access to network services" [18]. They propose a general implementation framework for client-side selection known as the Smart Client. This is similar to work by Bhattacharjee, et al. on application-layer Anycasting [22]. Both systems are flexible enough to incorporate different selection metrics, as are several distributed Web caching designs [1][3][23]. Consequently, all are complementary to this study and could benefit from the results.

The studies most similar to this paper are client-side selection studies by Carter and Crovella [20][21], Seshan, et al. [17], and Sayal, et al. [9]. Carter and Crovella use trace-driven simulation to compare HTTP transfer times for selection algorithms based upon geographical distance, hop count, `ping` probes, and an available bandwidth probe called `cprobe`. They find the fastest transfer times are obtained for the dynamic `ping` algorithm and for an algorithm that combines `ping` with `cprobe` data. Carter and Crovella conclude `cprobe` is unsuitable as a dynamic probe because of its high overhead, both in terms of time and network traffic. However, `cprobe` provides information similar to bandwidth estimators derived from historic data, so results in [20] augment findings for statistical bandwidth algorithms. In the SPAND project, Seshan, et al., passively collect bandwidth data for transfers from a server to all clients on the local LAN [17]. They show observed bandwidths are reasonably stable; that is, 90% of the observed bandwidths are within a factor of 4 of the predicted bandwidth. However, they do not report total time, nor do they compare their statistical bandwidth predictor to other metrics. Sayal, et al., directly compare performance of different selection metrics, but they use HTTP "request latency" as a cost function instead of response time [9]. Consequently, their results indicate only that past request latency is the best predictor of current request latency, and do not necessarily extrapolate to total response time. Further, [20] and [9] report metrics using means, and [17] uses bandwidth means and standard deviations for the selection criteria.

This paper expands upon the related studies by presenting results for different times-of-day and for clients on different regional networks, and by examining the relative costs of connection establishment, read latency, and bandwidth. We also describe the distributions of connection times, latencies, bandwidths, and total response time, and explain why median and semi-interquartile range (SIQR) are more appropriate statistics than mean and standard deviation for both statistical estimators and the performance metric.

III. SERVER SELECTION ALGORITHMS

From the client's perspective, user response time is the sum of DNS lookup, connection establishment, read latency, and remaining read time:

$$T = T_{DNS} + T_{Connect} + T_{Latency} + T_{Remaining} \quad (1)$$

where T_{DNS} is the DNS lookup time, $T_{Connect}$ is the time to establish a TCP/IP connection, $T_{Latency}$ is the time from sending the request to receiving the first packet of the reply, and $T_{Remaining}$ is the time to receive the remaining reply packets. DNS lookup exercises the DNS hierarchy and is independent of server load. It can be avoided if the client knows the server's IP address through caching or an independent metadata transfer mechanism [1][3]. For these reasons we ignore the DNS term. Connection time depends upon network latency and the wait in the server's application queue. Read latency includes the server delay incurred retrieving the object from disk, plus network delays due to transmission latency of the object's first data packet.

The remaining read time depends upon available bandwidth for the client-server connection and upon server load. Results in Section V show $T_{Remaining}$ dominates response time for moderate file sizes. However, all four components depend upon network conditions and, as we observed, can cause pathological behavior when packets are dropped by congested network routers. Using Eq. 1, the six selection algorithms in this experiment are defined as follows:

Random: Select a server randomly.

Latency: (statistical) Select the server with lowest median latency, $T_{Latency}$, in prior transfers.

BW: (statistical) Select the server with fastest median bandwidth in prior transfers, where bandwidth is

$$BW = \frac{bytes}{T_{Latency} + T_{Remaining}}. \quad (2)$$

Probe: (dynamic) Send dynamic probes to all servers and select the first to reply. Immediately request the object from that server without waiting for replies from other probes.

ProbeBW: (hybrid) Consider only n servers with the fastest median bandwidths. Send dynamic probes to these servers and select the first to reply. Immediately request the object from that server without waiting for replies from other probes. In our experiments $n = 3$.

ProbeBW2: (hybrid) Send a dynamic probe to all servers. Upon receiving the first probe reply, delay for half the reply time to see if any other servers respond almost as quickly. After the delay, select the server with the fastest median bandwidth among those who have replied.

IV. Methodology

The dynamic probe used in our experiments is `tcping`, our TCP/IP analogy of the well-known `ping` utility. Section IV-F describes `tcping` in more detail and explains its advantages over ICMP-based probes such as `ping`. Section IV also discusses the performance metric and file size normalization, overhead measurements, the client and server sets, and calibration data for the statistical latency and bandwidth estimators.

A. Performance metric and file size normalization

Client-side selection methods are designed for a set of heterogeneous, topologically-dispersed servers, whose response times depend upon both server and network effects. Even for homogeneous server sets with well-balanced load, response times can differ significantly because network routes between the client and the servers have different bandwidths and congestion patterns. Server load does not reflect route differences, therefore it is not an appropriate metric for client-side methods, although it can be used to compare server-side selection methods [8][14]. Conversely, high bandwidth connections do not guarantee fast

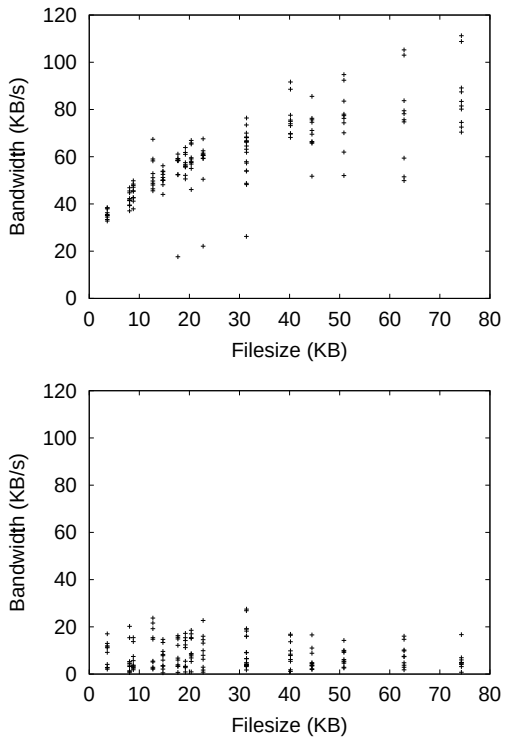

Fig. 1. Effect of file size on HTTP bandwidth. Graphs show HTTP GETs from `www.state.ca.us` to UTSA at night (top) and during the day (bottom).

response times if the server has a long queue or slow file system. The appropriate metric for client-side selection algorithms is user response time because it reflects both server and network effects. Unfortunately, response times cannot be compared directly if objects are of different sizes. Other researchers approach this problem in several ways. Karaul, et al., [19] measure response times but restrict their server set to 10 mirror sites with identical objects. In the SPAND project [17], the performance evaluation includes data for many HTTP servers and different object sizes, but uses bandwidth rather than response time as the metric. Sayal, et al., [9], also use an unrestricted server set, but compare selection algorithms on the basis of latency and disregard bandwidth. Rather than restricting either the server set or the metric, our solution is to normalize response time.

Let T be the measured response time for retrieving an object of size N. Let $\widehat{T}$ be the computed normalized time to retrieve the object if it were of size $\widehat{N}$. To normalize the response time, we measure T and its components $T_{Connect}$, $T_{Latency}$, and $T_{Remaining}$. We also record the total number of bytes, N, and number of bytes in the first read, $N_{Latency}$. Assuming the bit rate for remaining reads is independent of file size, the response time normalized to reference size $\widehat{N}$ is:

$$\widehat{T} = T_{Connect} + T_{Latency} + T_{Remaining}\left(\frac{\widehat{N} - N_{Latency}}{N - N_{Latency}}\right) \quad (3)$$

How valid is the assumption that post-latency bandwidth is independent of file size? In the absence of traffic congestion, the sliding window and slow start mechanisms of TCP/IP tend

to produce higher bandwidths for larger objects. Under congested conditions, the packet drops and timer expirations reduce this effect. To test the validity of this assumption, we measured download times for various files between individual client-server pairs. Fig. 1 plots bandwidth for remaining read packets as a function of file size. Although the graphs show only one client-server pair, they are representative of other measured pairs. During nights and weekends when the network is quiescent, bandwidth tends to increase with file size (Fig. 1, top). However, during the day when the network is busy, bandwidth appears independent of file size (Fig. 1, bottom). This indicates normalization is valid for busy periods, but it must be restricted to small size ranges for quiescent periods. Fortunately, we are most interested in high traffic periods where there is the most need for performance improvement.

B. Overhead

Dynamic and hybrid algorithms include network communication, therefore they are likely to have higher overhead than statistical algorithms which rely solely upon local table lookup. To insure fair comparisons, the overhead is measured and added to the normalized time:

$$\widehat{T}_{Total} = \widehat{T} + T_{Overhead}. \quad (4)$$

C. Clients and servers

The server selection experiments were run concurrently on clients at three Texas universities (see Table II). Although clients are located in the same state, they are in different cities, are connected to different regional networks, and use different primary Internet backbones. All have T1 or better connections to the Internet and accessed the same set of servers in the experiment. To model a widespread caching or replication system, the server set needs to be distributed across the country and across Internet backbones. Also, servers should be stable and reasonably well-connected. To fulfill these conditions, we constructed the server set from official World Wide Web servers for state governments in the United States, `www.state.**.us`, where `**` denotes the state postal abbreviation. We also included two special servers that are near the clients, both geographically and topologically, and which have fast routes to the clients. By dividing servers into national and nearby subgroups, we can vary the probability of having the object at a nearby server and observe how different selection algorithms respond. Fig. 2 shows the locations of the three clients and 42 servers used in the study. A few states were omitted because the test object moved during the calibration or experiment measurements, or because the server did not respond to `tcping` probes (see Section IV-F).

D. Measurement sessions

A measurement session begins by randomly picking ten unique candidates from the pool of 42 servers. The candidate set models a group of replicated mirror sites or cache servers which have copies of the desired object. As regional servers might return objects more quickly than most national servers,

Fig. 2. Location of servers (S) and clients (C).

TABLE II
CLIENTS: TEXAS A&M (A&M), UNIVERSITY OF HOUSTON (UH), AND UNIVERSITY OF TEXAS AT SAN ANTONIO (UTSA).

	A&M	**UH**	**UTSA**
Regional network	tx-bb.net	verio.net	the.net
Primary backbone	BBN Planet	Verio	Sprint
High perf. network	vBNS	vBNS	none
Operating system	Solaris 2.6	Solaris 2.6	Solaris 2.6 Linux 2.0.30
Calibration period	12/98 - 3/99	12/98 - 2/99	12/98 - 3/99
Calibration meas.	3309	3839	7178
Experiment period	3/99 - 5/99	3/99 - 5/99	3/99 - 5/99
Experiment sessions	1630	1710	1720
Image Day	290	276	242
Image Night	914	852	907
HTML Day	110	159	86
HTML Night	316	423	485
Connect failure rate			
Day	0.009	0.008	0.008
Night	0.007	0.008	0.008
Read failure rate			
Day	0.003	0.003	0.025
Night	0.001	0.001	0.001

the probability of including one or more regional servers was fixed at 25% for all experiments except those which explicitly investigated the effect of nearby servers. This was accomplished by using different selection probabilities for regional and national servers. Within a session, each selection algorithm chooses a server from the candidate set, and the program immediately downloads the object from that server, recording number of bytes, algorithm overhead, connection time, read latency, and remaining read time. A session loops over the six selection algorithms in random order, pausing three minutes between algorithm measurements to avoid creating network congestion. One problem we constantly encounter is the high variability of repeated measurements. In many cases we observed that when more than one algorithm within a session selects the same server, the measured download times often differ substantially, with no obvious pattern. In order to fairly compare algorithms and eliminate this noise, the analysis uses the first session download time for a server for all algorithms in the session that select that server.

The experiment consisted of over 1600 measurement sessions spanning a five week period. Table II lists the number of measurement sessions for each client under each experimental condition, together with failure rates due to connection refusals, connection timeouts, and read timeouts. If the download for any algorithm within a session failed, the entire session was excluded from the analysis. This had little effect, as connection and read requests failed infrequently. Connection failures occur in 0.7% to 0.9% of the measurements, and are independent of both client and time-of-day. Read failure rates are typically even smaller, ranging from 0.1% to 0.3%, except during the day at UTSA where the rate was 2.5%.

E. Determining bandwidth and latency predictors

Statistical selection algorithms require historic data to determine performance estimators such as latency or bandwidth medians. In this experiment, we collected calibration data prior to running the selection experiment, rather than continuously updating the estimators during the experiment. The calibration data consist of several thousand HTTP downloads for the same client-server pairs used in the selection experiment. The distributions of latency and bandwidth for individual client-server pairs are highly skewed, with extremely long, flat tails and exhibit large variability. Because of the skew, we use the median latencies and bandwidths, rather than the means, as statistical estimators [24]. Both time-of-day and file size effects are apparent in the calibration data, and, as Fig. 1 shows, these factors interact. This suggests that different statistical estimators be computed for different times of day and file size combinations. For this experiment, we used two time-of-day categories, (**Day** and **Night**), and two file size categories (**Image** and **HTML**). Combining categories creates four experimental conditions: **Image–Day**, **Image–Night**, **HTML–Day**, and **HTML–Night**. The **Day** category contains data for Monday through Friday, 9 am to 5 pm, and the **Night** category contains all other times. HTML files are small text files, normalized to 6 KB. Image files are larger GIF and JPG files, normalized to 50 KB. The calibration and selection experiment used one HTML and one image file from each server. Median latencies and bandwidths for each client-server pair under each of the four conditions were computed from the calibration data. In the selection experiment, the statistical and hybrid algorithms check time-of-day and object type, then use the appropriate estimator for current conditions.

F. `tcping`

For dynamic algorithms we use our own probe, `tcping`, which sends a TCP SYN packet to an unused port on the server and listens for a TCP RST reply. This involves a single round trip exchange of TCP header packets. By contrast, the standard `ping` utility uses the ICMP protocol to send an ECHO_REQUEST datagram to the server's echo port and listens for the ECHO_RESPONSE [25]. Because ICMP datagrams are sent over raw sockets, `ping` requires `root` privilege to execute. On UNIX systems, this permission is granted to users by setting the access mode of the executable file. Using TCP/IP instead of ICMP gives `tcping` the following advantages:

1. `tcping` is called as a user function. Conversely, users programs call `ping` by invoking a `system()` kernel call, which forks a new execution shell and incurs context switching overhead.
2. A user program can send concurrent `tcping` probes to multiple servers and monitor their replies, allowing clients to run dynamic selection algorithms without `root` access or modification of the server.
3. `tcping` exercises a portion of the TCP protocol stack, therefore it includes an element of TCP/IP performance in its measurement.

For security reasons, some servers or their firewalls do not send ECHO_RESPONSE datagrams, rendering `ping` useless. Likewise, some may not send TCP RST replies for unused ports, which renders `tcping` useless. Of the 50 `www.state.**.us` servers, five did not respond to `ping` and five did not respond to `tcping`. Two cases overlapped; that is, two servers did not respond to either `ping` or `tcping`. One method for combining security features with dynamic probing would be to install a probe port on the server. A server probe port could also return information about current load on the application server, thus providing a simple, low-cost method for including load balancing in client-side selection methods.

V. Results

For each algorithm, the response time distribution exhibits large skew, long, flat tails, and high variability. This is not surprising, as these are the same characteristics observed in the calibration data for repeated measurements of a single object, and are consistent with results of other researchers [26]. As with latency and bandwidth estimators, the skew and outliers in these distributions make median the preferred index of central tendency and semi-interquartile range (SIQR) the preferred index of dispersion [24].[1] Mean and standard deviation are inappropriate for response time because they are highly sensitive to the extreme outliers found in response time distributions. Fig. 3 compares performance of the selection algorithms at the three client sites using total response time $\widehat{T}_{Total}$. These bar graphs show results for the four file size and time-of-day categories. Table III reports medians and SIQR statistics for total response time and for its component terms. Because both typical response time and its variability are important quality-of-service issues to the client, it is important to note which algorithms have low median and low SIQR values. From the bar graphs and Table III, we observe the following:

- Probing is better than statistical algorithms at reducing response time:

$$Random > Latency > BW \approx ProbeBW2 > ProbeBW \approx Probe$$
$$slowest \longrightarrow fastest$$

- All algorithms have low overhead.

[1] SIQR is one-half the difference between the 75% and 25% percentiles.

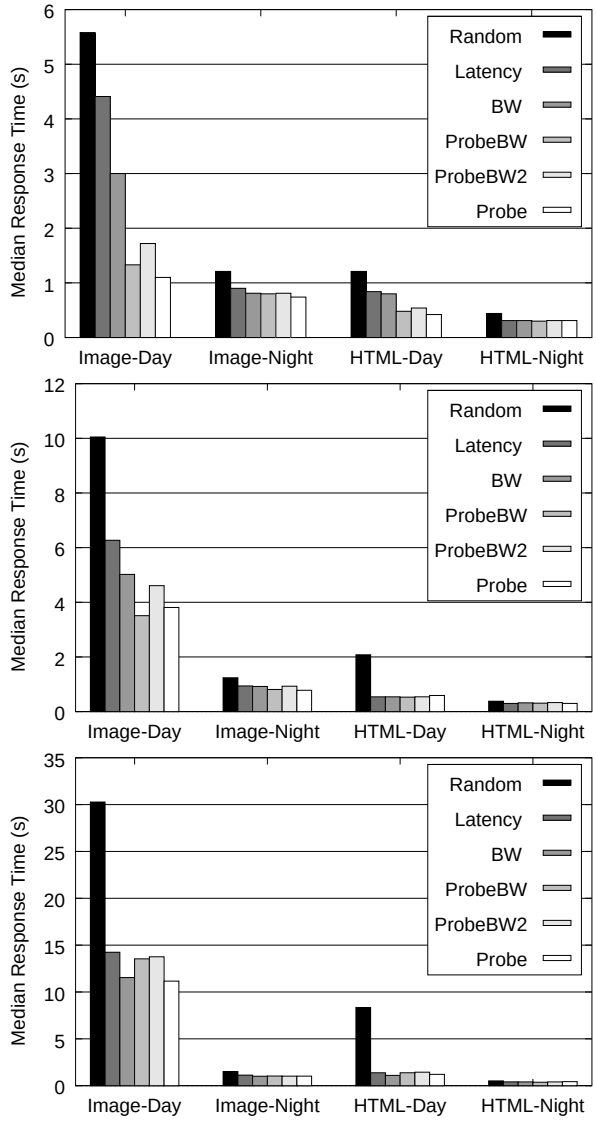

Fig. 3. Median response time, $\widehat{T}_{Total}$, for the server selection algorithms at A&M (top), UH (middle), and UTSA (bottom).

- When the network is quiescent, there is little difference between algorithms.
- *Random* has the highest variability and, in most cases, *Probe* has the lowest.
- *Latency* performs worse than any algorithm except *Random*.
- The hybrid algorithms, *ProbeBW* and *ProbeBW2*, do not improve upon the simple probe algorithm.
- *Probe* is markedly superior to *BW* at A&M, but less so at the other two clients. We explain this in Section V-D.

A. Overhead, connection, latency, and read times

Table III reports medians of elapsed time, in seconds, for algorithm overhead, connection establishment, and read time normalized to a 50 KB reference size. The normalized read time is

TABLE III
RESPONSE TIMES (SEC) FOR $\widehat{N}$ = 50KB, M-F 9AM-5PM.

Algorithm	$T_{Overhead}$ *med*	$T_{Connect}$ *med*	*SIQR*	$\widehat{T}_{Read}$ *med*	*SIQR*	$\widehat{T}_{Total}$ *med*	*SIQR*
A&M							
Random	0.00	0.2	0.2	5.2	7.6	5.6	8.3
Latency	0.00	0.1	0.1	3.5	4.4	4.4	5.3
BW	0.02	0.1	0.1	2.3	3.2	3.0	3.5
ProbeBW	0.08	0.1	0.0	1.1	2.0	1.3	2.0
ProbeBW2	0.06	0.1	0.0	1.4	2.1	3.7	2.2
Probe	0.08	0.1	0.0	0.9	1.2	1.1	1.4
UH							
Random	0.00	0.1	0.1	9.6	10.3	10.1	10.7
Latency	0.00	0.1	0.0	5.9	5.2	6.3	5.3
BW	0.02	0.1	0.0	4.4	4.4	5.0	4.6
ProbeBW	0.07	0.1	0.0	2.3	5.6	3.5	5.8
ProbeBW2	0.05	0.1	0.0	3.7	5.4	4.6	5.8
Probe	0.06	0.1	0.0	2.9	5.9	3.8	5.9
UTSA							
Random	0.00	0.7	0.8	28.4	25.9	30.3	26.7
Latency	0.00	0.2	0.3	12.6	13.8	14.2	15.5
BW	0.00	0.2	0.2	10.7	11.4	11.5	12.3
ProbeBW	0.20	0.2	0.1	12.5	12.4	13.5	13.4
ProbeBW2	0.17	0.2	0.1	12.7	12.4	13.8	13.0
Probe	0.19	0.2	0.0	10.7	10.6	11.2	11.2

the measured latency plus the normalized remaining read time as defined in Eq. 3:

$$\widehat{T}_{Read} \;=\; T_{Latency} + T_{Remaining}\left(\frac{\widehat{N} - N_{Latency}}{N - N_{Latency}}\right) \qquad (5)$$

Table IV displays the relative fractions of total time, $\widehat{T}_{Total}$, attributable to overhead, connection, latency, and normalized remaining read time. These fractions were obtained by computing fractions for each measurement and taking medians over all measurements. The smallest contribution comes from overhead. Overhead for statistical algorithms is insignificant because they rely upon local table lookup. Overhead for dynamic and hybrid algorithms using `tcping` are only slightly higher, with medians ranging from 20 to 70 ms at the two better-connected clients, and from 40 to 200 ms at the other client. Interestingly, connect, latency, and remaining read times tend to follow the same algorithm performance order as total time. Table IV shows that, after correcting for overhead, their relative contributions are nearly constant across algorithms. For example, selecting servers randomly tends to result in longer connect times than those for the other five selection algorithms. *Random* also produced the most pathologically long connect times, while *Probe* produced the fewest: 29 of the 5060 measured downloads for *Random* had connect times of over ten seconds, as compared to 21 for *Latency*, 18 for *BW*, and 6 for *Probe*. The same patterns hold for latency and normalized read time.

Although the median fractions in Table IV are relatively constant, all components except $T_{Overhead}$ contribute outliers whose absolute values far exceed the median total response time.[2] For example, several connection times took between 70

[2]The number and magnitude of outliers are limited by timeout values. Our experiments use a 5 second timeout for `tcping`, 90 seconds for connect, and 10 minutes for read.

TABLE IV
FRACTION OF TOTAL RESPONSE TIME (MEDIANS).

Algorithm	*Fraction of total time*				*Fraction ignoring Overhead*		
	$f_{Overhead}$	$f_{Connect}$	$f_{Latency}$	$f_{Remaining}$	$f_{Connect}$	$f_{Latency}$	$f_{Remaining}$
Image - Day							
Random	0.00	0.03	0.04	0.90	0.03	0.04	0.90
Latency	0.00	0.03	0.04	0.90	0.03	0.04	0.90
BW	0.00	0.03	0.05	0.88	0.03	0.05	0.88
ProbeBW	0.03	0.03	0.06	0.81	0.03	0.07	0.86
ProbeBW2	0.06	0.03	0.06	0.78	0.04	0.07	0.86
Probe	0.03	0.03	0.07	0.80	0.03	0.07	0.86
Image - Night							
Random	0.00	0.07	0.09	0.84	0.07	0.09	0.84
Latency	0.00	0.07	0.08	0.85	0.07	0.08	0.85
BW	0.00	0.07	0.10	0.82	0.07	0.10	0.82
ProbeBW	0.06	0.06	0.09	0.77	0.06	0.10	0.83
ProbeBW2	0.12	0.06	0.09	0.71	0.07	0.10	0.82
Probe	0.07	0.05	0.08	0.77	0.06	0.09	0.84

and 90 seconds, as compared to most median total times of under 20 seconds. In general, we find distributions of connect time, latency, and remaining read time exhibit the same large skews and long, low, flat tails as $\widehat{T}_{Total}$. We attribute the skews and long tails to dropped packets and the resulting TCP/IP retransmissions and window adjustments.

B. Time-of-day effects

Differences between server selection algorithms are far more pronounced during busy daytime hours than for quieter night and weekend periods. This reflects both the effect of network load on the algorithms, and the periodicity of Internet traffic load. Internet traffic and server loads increase in the mornings, decrease in late afternoon, and are much lower on weekends than during the week. To simplify the analysis, we separate data into only two time categories, **Day** and **Night**, choosing boundary hours of 9 am and 5 pm CST. The cutoff hours were chosen by plotting medians of $\widehat{T}_{Total}$ *vs.* time-of-day for weekday data, and observing the hours where $\widehat{T}_{Total}$ rises and falls sharply (see Fig. 4). In the remainder of this paper, we concentrate on results for the larger **Image** files during **Day** hours, where waits are longer and performance improvement is most needed. The two hybrid algorithms always performed worse than *Probe* and better than *Latency*. These are omitted from Fig. 4 for the sake of clarity.

Hourly plots of elapsed time point out an important fact: the performances of the selection algorithm maintain their relative order as the load changes. With minor exceptions, the curves do not cross, implying that as load changes clients need not use different algorithms to achieve optimal performance.

C. Pathological delays

One goal of server selection algorithms is to significantly reduce the number of pathologically long wait times. As noted in Section V-A, connect time, latency, and remaining read time can each be responsible for pathologically long delays. Many long delays cannot be predicted by past connection, latency, or bandwidth measurements, nor by `tcping` probes sent imme-

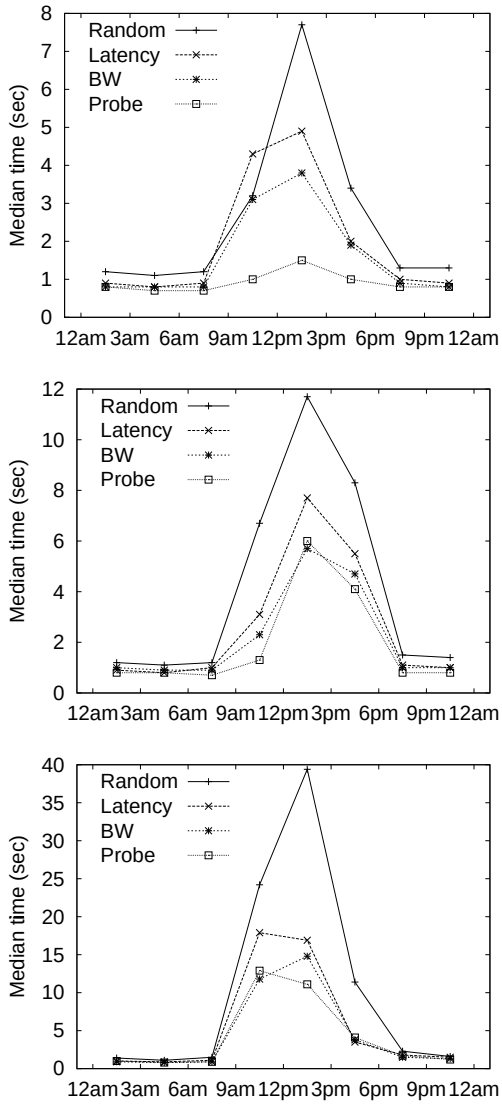

Fig. 4. Effect of time-of-day on the selection algorithms for **Image** files at A&M (top), UH (middle), and UTSA (bottom).

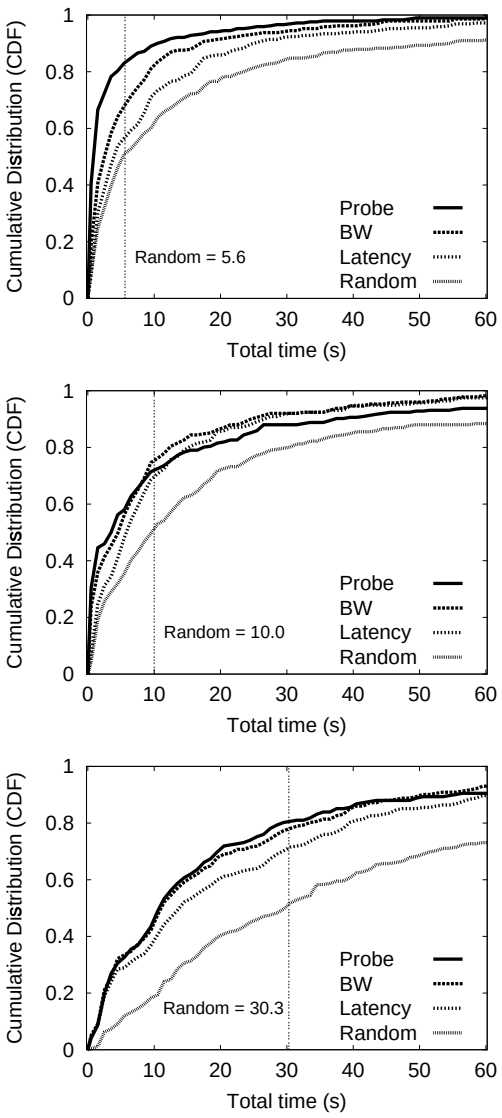

Fig. 5. Cumulative distributions of $\widehat{T}_{Total}$ for **Image** files ($\widehat{N} = 50KB$), M-F 9am to 5pm, at A&M (top), UH (middle), and UTSA (bottom).

diately prior to the download. In rare cases, the server with the lowest historic latency, highest previous bandwidth, and fastest `tcping` probe timed out on a connection request, or took tens of minutes rather than seconds to deliver the file. To determine which algorithms best reduce pathological behavior, we plot the cumulative distribution function (CDF) of total response times for each algorithm (Fig. 5). Once again we find the same ordering of algorithm performance: *Probe* and *BW* typically produce the fewest pathological cases, *Random* produces the most, and *Latency* turns in an intermediate performance. The CDF curves for hybrids *ProbeBW* and *ProbeBW2* are close to that for *Probe* and are again omitted for clarity.

CDF curves also allow us to compare performance distributions at clients with different connection bandwidths. We compare across clients by drawing a vertical line to mark the median time for *Random*, then reading cumulative distribution values for other algorithms at that client. Alternatively, we can expand or contract the horizontal axis according to the ratios of *Random* medians. A&M's median time for *Random* is approximately half that of UH, and one-sixth that of UTSA. When we scale the time axes by these ratios, the CDF curves have similar shapes, indicating the algorithms have similar performance distributions at different clients. The distribution tails of the algorithms are not noticeably longer, lower, or flatter at more poorly connected clients, Consequently, if an algorithm produces few pathological outliers at one client, it will likely produces few pathological outliers at all clients.

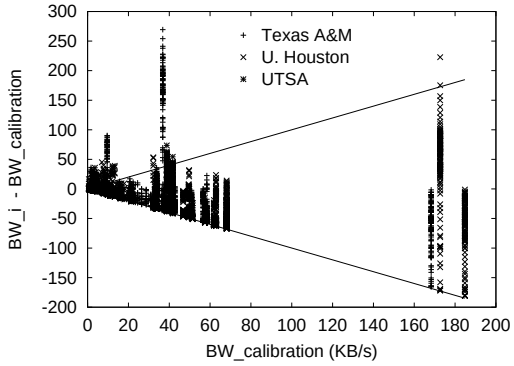

Fig. 6. Bandwidth errors for individual measurements.

D. Effect of network modifications

The effectiveness of statistical algorithms such as *Latency* and *BW* is limited by the accuracy of their calibration data. If the network connection between a client and server is upgraded, statistical algorithms will underperform until their calibration data are updated. This problem is illustrated in the results for A&M. Subsequent to calibration, a high speed connection was established between A&M and the University of Texas Medical Center in Houston, a member of the server set. A second, less dramatic improvement was made to a connection between A&M and another server in the set. Because *BW* and *Latency* relied upon stale bandwidth and latency data, they did not choose these two servers as often as they should have, and consequently did not perform as well at A&M as they did at the other two clients. The upgraded servers stand out when bandwidth errors are plotted against the calibration bandwidth, as in Fig. 6. In this graph, bandwidth error for the i^{th} measurement is given by $Error_i = BW_i - BW_{calibration}$. Data points above the upper diagonal line have a measured bandwidth over twice the calibration bandwidth. At 38 KB/s, there is a vertical smear of points well above this upper line, and a shorter series of points at 9 KB/s. These points flag the two upgraded connections, whose identity was easily determined by referring to calibration data.

E. Importance of nearby servers

In this section, we investigate the importance of locating caches or other services at topologically nearby servers with relatively fast network connections to the client. A topologically nearby connection does not go through an Internet backbone

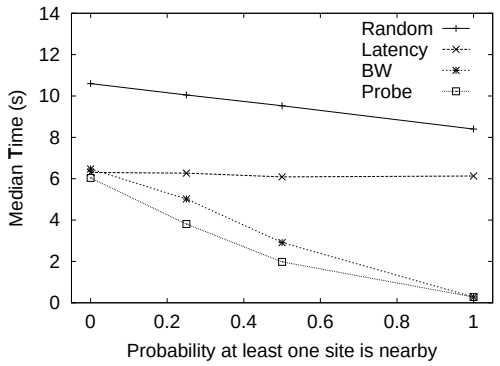

Fig. 7. Effect of including nearby servers at UH.

or backbone exchange point. The server set includes two well-connected servers located in the same state as the clients: the University of Texas at Austin and the University of Texas Medical Center in Houston. All three clients connect to these nearby servers through either regional networks or vBNS. As explained in Section IV-D, each session has a 25% probability that its candidate set includes at least one nearby server, or $P = 0.25$. How might results change if we had chosen a different probability? To answer this question, we examined session data to see if a session includes at least one nearby server and, if so, included that session with probability P in the analysis. Fig. 7 shows results for $P = 0$, 0.25, 0.50, and 1. $P = 0$ models the case where no nearby servers have a copy of the desired object, and $P = 1$ models the case where at least one nearby server has a copy. The ordering for median total times

$$Random > Latency > BW \geq Probe$$

holds across probabilities. For two clients, the *Probe* and *BW* curves drop more steeply, while *Latency* has an almost flat response, indicating *Probe* and *BW* are more efficient at finding nearby servers. At $P = 1$, the median total times for *Probe* and *BW* fall to well under a second at two clients, and under 3 seconds at the third client. Compared to random selection with no nearby servers, this experiment obtained an order-of-magnitude speedup by including at least one nearby server and using *Probe* or *BW*.

VI. Conclusions and future work

- **Use dynamic probes instead of statistical estimators to select servers.**

The principal conclusion of this study is that client-side selection algorithms should use simple network probes to select servers, instead of past performance data such as latencies or bandwidths. *Probe* clearly outperforms *Latency*, and performs as well or better than *BW* under all conditions. Dynamic probes are easy to implement, adapt automatically to changes in the network or server, and add little traffic or delay overhead.

Conversely, these experiments highlight several disadvantages of statistical algorithms. Performance of algorithms that rely on statistical estimators is limited by the accuracy and variability of the calibration data. Good statistical estimators for Internet bandwidths and latencies are difficult and cumbersome to obtain because 1) clients must collect data for each potential server, 2) bandwidth and latency depend upon time-of-day and day-of-week, 3) repeated measurements exhibit large skew, long tails, and high variability, and 4) estimators go stale when network or server resources are upgraded. For true distributed caching or in mobile networks, the candidate set may be large and dynamic, making it difficult to collect and maintain statistical data for each potential server. Time-of-day dependence forces clients to repeat calibration measurements during each time category for each server. Bandwidth may also depend upon file size, necessitating more measurement categories. If measured data were consistent and stable, this would not be a problem. Unfortunately, the skew and variability of latency and bandwidth distributions require clients to collect many repeated measurements for each server under each condition. Even then, statistical bandwidth and latency algorithms often do not select a fast server because these distributions have large spreads: the more server distributions overlap, the poorer the predictability of the estimator. Because large variability is an inherent feature of Internet latencies and bandwidths, it imposes a hard limit on the performance of statistical algorithms. Finally, statistical algorithms have difficulty adapting to both outages and upgrades in the server or network. We saw the consequences in our results for A&M, where stale calibration data for two upgraded connections noticeably degrade performance. Stale data can be avoided either by periodic recalibration or by continuously collecting data from actual downloads and computing median values over a sliding window. Both methods have problems. Periodic recalibration adds substantial network traffic, while data from actual downloads are slow to discover upgrades because statistical algorithms tend to avoid servers with poor past performance.

- **Include nearby servers in the candidate set.**

All the selection algorithms, including random selection, perform better when a topologically nearby server is included in the candidate set. For HTTP downloads during the day, we achieve speedups of 8 to 35 by using either *Probe* or *BW* and including at least one nearby server, as compared to randomly selecting a server from a candidate set that did not include any nearby servers.

- **Use total elapsed time rather than latency or server load to evaluate performance.**

In terms of performance, the goal of distributed caching and replication is to reduce user response time. Server load is not an appropriate metric for client-side selection algorithms because it does not reflect differences in network transmission times. When latency is used both as the estimator and as the metric [9], results prove only that prior latency is a good predictor of current latency. The results of this study show prior latency is not as successful as either prior bandwidth or network probes at predicting total response time.

- **Pathologically long delays can be reduced but not eliminated by these algorithms.**

None of the algorithms eliminates pathologically long delays; however, all algorithms reduce the number of such delays by 7% to 20% over random selection. Pathological behavior is difficult to predict because it appears in each component of total time: connect, latency and read distributions all exhibit the same large skew and long, flat tails that characterize total time. Statistical algorithms reduce the number of long delays because some client-server connections statistically experience fewer long delays. Network probes reduce pathological behavior because they discover current congestion problems. Neither method is especially successful. We find connection, latency and read times can be independently responsible for pathological behavior; that is, a long delay for one component does not necessarily imply a long delay for any other component. This is another reason to include all components in the evaluation metric, even though one term may dominate the median value.

- **Use bandwidth data to reduce the number of probes.**

When the number of potential servers is large, we do not wish to send dynamic probes to every server. The hybrid algorithm *ProbeBW* filters out servers with historically poor bandwidth, then sends probes to the remaining servers. Our results show this hybrid method performs almost as well as probing all servers, and, unlike *BW*, responds robustly to network upgrades and poor calibration data. This robust behavior makes it feasible to compute calibration data from a sliding window of prior downloads. Consequently, *ProbeBW* offers a viable method for reducing the number of dynamic probes for large numbers of candidate servers.

- **Would a server load probe improve performance?**

Statistical algorithms reflect past network and server loads, but offer no information about current conditions. Network probes reflect current network load, but provide no information about current server load. If the server is overloaded with respect to its CPU or disk subsystem, the delay is unrelated to network congestion. More research is needed to determine how often such situations occur and whether they can be predicted using a dynamic server load probe. One approach to a server load probe would be to install an information port on the server. The HTTP server process would periodically post current load information to the load port through shared memory or other low-cost mechanism. Operating independently of the HTTP process, the information port would reply to load queries with a small UDP packet. (Alternatively, Fei, et al., suggest the server push its load information to client agents [27].) Implementing such an information port and load probes is easy. The difficultly lies in testing whether server load probes improve server selection for a representative set of Web servers, which currently provide no load data to their clients. Studies separating network and server effects are an important area of future research.

ACKNOWLEDGEMENTS

The authors wish to thank Steve Robbins for his help in preparing this paper.

REFERENCES

[1] S. G. Dykes, C. L. Jeffery, and S. Das, "Taxonomy and design analysis for distributed web caching," in *Proc. of the IEEE Hawaii Int'l. Conf. on System Sciences (HICSS'99)*, Jan. 1999.

[2] National Laboratory for Applied Network Research (NLANR), "Global caching hierarchy," http://www.nlanr.net/.

[3] R. Tewari, M. Dahlin, H. M. Vin, and J. S. Kay, "Design considerations for distributed caching on the Internet," in *Proc. of the Int'l. Conf. on Distributed Computing Systems (ICDS'99)*, 1999.

[4] L. Zhang, S. Floyd, and V. Jacobson, "Adaptive web caching," in *Proc. of the 2nd Web Cache Workshop*, June 1997.

[5] H. Balakrishnam, S. Seshan, M. Stemm, and R. H. Katz, "Analyzing stability in wide-area network performance," in *Proc. of ACM SIGMETRICS Conf. on Meas. & Modeling of Computer Systems*, June 1997.

[6] M. E. Crovella and A. Bestavros, "Self-similarity in world wide web traffic evidence and possible causes," in *Proc. of the 1996 ACM SIGMETRICS Int'l. Conf. on Meas. and Modeling of Computer Systems*, May 1996.

[7] J. Mogul, "Network behavior of a busy web server and its clients," Tech. Rep. Research Report 95/5, Digital Equipment Corporation, Oct. 1995.

[8] D. Andresen, T. Yang, V. Holmedahl, and O. H. Ibarra, "SWEB: Towards a scalable World Wide Web server on multicomputers," in *Proc. of the 10th Int'l. Parallel Processing Symp. (IPPS'96)*, Apr. 1996.

[9] M. Sayal, Y. Breitbart, P. Scheuermann, and R. Vingralek, "Selection algorithms for replicated web servers," *Performance Evaluation Review*, vol. 26, no. 3, pp. 44–50, Dec. 1998.

[10] D. Johnson and S. Deering, "RFC2526: Reserved IPv6 subnet Anycast addresses," Mar. 1999.

[11] C. Partridge, T. Mendez, and W. Milliken, "RFC1546: Host Anycasting service," Nov. 1993.

[12] T. Brisco, "RFC1794: DNS support for load balancing," Apr. 1995.

[13] E. D. Katz, M. Butler, and R. McGrath, "A scalable HTTP server; the NCSA prototype," *Computer Networks and ISDN Systems*, vol. 27, pp. 155–164, 1994.

[14] D. M. Dias, W. Kish, R. Mukherjee, and R. Tewari, "A scalable and highly available web server," in *Proc. of the 41st IEEE Computer Society Int'l. Conf.*, Feb. 1996.

[15] J. Gwertzman and M. Seltzer, "The case for geographical push-caching," in *Proc. of the 1995 Workshop on Hot Operating Systems (HotOS-V)*, 1995, pp. 51–55.

[16] D. Wessels and K. Claffy, "RFC 2187: Application of internet cache protocol (ICP), version 2," Sep. 1997.

[17] S. Seshan, M. Stemm, and R. H. Katz, "SPAND: Shared passive network performance discovery," in *Proc. of the First USENIX Symp. on Internet Technologies and Systems (USITS'97)*, Dec. 1997.

[18] C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson, and D. Culler, "Using smart clients to build scalable services," in *Proc. of the 1997 USENIX Annual Tech. Conf. (USENIX'97)*, 1997, pp. 105–117.

[19] M. Karaul, Y. A. Korilis, and A. Orda, "A market-based architecture for management of geographically dispersed, replicated web servers," in *Proc. of the First Int'l. Conf. on Information and Computation Economics (ICE'98)*, 1998, pp. 158–165.

[20] R. L. Carter and M. E. Crovella, "Server selection using dynamic path characterization in wide-area networks," in *Proc. of IEEE Infocom'97*, Apr. 1997.

[21] M. E. Crovella and R. L. Carter, "Dynamic server selection in the internet," in *Proc. of the Third IEEE Workshop on the Architecture and Implementation of High Perf. Comm. Subsystems (HPCS'95)*, Aug. 1995, pp. 158–162.

[22] S. Bhattacharjee, M. H. Ammar, E. W. Zegura, V. Shah, and Z. Fei, "Application-layer Anycasting," in *Proc. of IEEE Infocom'97*, Apr. 1997.

[23] Squid Internet Object Cache, http://squid.nlanr.net/.

[24] R. Jain, *The Art of Computer Systems Performance Analysis*, John Wiley & Sons, Inc., New York, NY, 1991.

[25] W. R. Stevens, *UNIX Network Programming, Volume 1*, Prentice Hall PTR, Upper Saddle River, NJ, second edition, 1998.

[26] A. Myers, P. Dinda, and H. Zhang, "Performance characteristics of mirror servers on the Internet," in *Proc. of IEEE Infocom'99*, Mar. 1999.

[27] Z. Fei, S. Bhattacharjee, E. W. Zegura, and M. H. Ammar, "A novel server selection technique for improving the response time of a replicated service," in *Proc. of IEEE Infocom'98*, Mar. 1998.