
ELSEVIER

Computer Networks 31 (1999) 2529–2558

COMPUTER NETWORKS

www.elsevier.com/locate/comnet

On the network impact of dynamic server selection

Robert L. Carter [a,*,1], Mark E. Crovella [b]

[a] *Telcordia Technologies, 445 South Street, Morristown, NJ 07960, USA*
[b] *Computer Science Department, Boston University, 111 Cummington Street, Boston, MA 02215, USA*

Abstract

Widespread replication of information can ameliorate the problem of server overloading but raises the allied question of server selection. Clients may be assigned to a replica in a static manner or they may choose among replicas based on client-initiated measurements. The latter technique, called dynamic server selection (DSS), can provide significantly improved response time to users when compared with static server assignment policies (for example, based on network distance in hops).

In the first part of this paper we demonstrate the idea of DSS using experiments performed in the Internet. We compare a range of policies for DSS and show that obtaining additional information about servers and paths in the Internet before choosing a server improves response time significantly. The best policy we examine adopts a strategy of never adding more than 1% additional traffic to the network, and is still able to provide nearly all the benefits of the most expensive policies.

While these results suggest that DSS is beneficial from the network user's standpoint, the system-wide effects of DSS schemes should also be closely examined. In the second part of this paper we use large-scale simulation to study the system-wide network impact of dynamic server selection. We use a simulated network of over 100 hosts that allows local-area effects to be distinguished from wide-area effects within traffic patterns.

In this environment we compare DSS with static server selection schemes and confirm that client benefits remain even when many use DSS simultaneously. Importantly, we also show that DSS confers system-wide benefits from the network standpoint, as compared to static server selection. First, overall data traffic volume in the network is reduced, since DSS tends to diminish network congestion. Second, traffic distribution improves – traffic is shifted from the backbone to regional and local networks.

Keywords: Distributed systems; Internet and network algorithms; Replication; Server selection; Network simulation

1. Introduction

The server selection problem arises in client–server systems when clients are given a choice among several functionally equivalent sources of service. Services such as NNTP, DNS, distributed databases, and the WWW all use replication in various forms to achieve scalability, and their clients may benefit by choosing among the replicas. As more such services are deployed, the server selection problem becomes more important.

In many cases, a client is assigned to a server in a static manner. For example, the client may be assigned to the server that is the fewest network hops away. Replica placement using historical

* Corresponding author. Tel.: +1-973-829-4371; fax: +1-973-829-2645.

E-mail addresses: carter@research.telcordia.com, carter@bu.edu (R.L. Carter), crovella@cs.bu.edu (M.E. Crovella).

[1] This work was done while the author was at Boston University.

1389-1286/99/$ - see front matter
PII: S1389-1286(99)00119-X

demand patterns to determine where copies should be located [12] may have an associated static server assignment scheme. Alternatively, a client may take advantage of measurements of the system state when making a choice among servers. Among the metrics that could be used are: load at the server, recency of information, network traffic volume, and distribution of traffic. In particular, a WWW user may desire to minimize latency when accessing a certain page. Viewed from the server side, a set of servers may cooperate to achieve load balancing by re-directing requests for service to less loaded replicas. These *dynamic* server selection schemes are measurement-based and therefore adaptive to current conditions.

In this paper we report on tools and techniques for selecting among information servers without requiring 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. While we focus on the problem of dynamically selecting a server for replicated documents, most of our results should be applicable to replicated services in general.

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. Both of these metrics are measures of distance in a network and it is intuitively appealing to suppose that minimizing distance is the correct approach to server selection. Surprisingly, we show that these two metrics yield very different results in practice. We then present evidence that dynamic server selection (DSS) 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 have in prior work 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. Full details appear in [7]. In this work, we show how to use CPROBE and its available bandwidth measurements to improve server selection. We report on experiments based on measurements in the operational Internet which suggest that DSS can reduce response time for clients.

While the empirical results from the live Internet experiments are encouraging, there remain many questions which cannot be answered through experiments in physical networks due to limited resources and lack of control over the network. For example, the live experiments indicate that a single client practicing DSS should see improvement in response time. However, it is not clear that many clients simultaneously using DSS would still perform well. Before suggesting their widespread use in live networks it is important that we determine whether their benefits will continue to apply in such an environment. Wide-area network effects are also hard to observe empirically. However, these kinds of measurements can easily be made on a simulated network. In order to answer questions regarding the deployment of the measurement tools and the DSS protocols previously developed, we have undertaken a large-scale simulation.

Thus, the questions we studied through simulation are:

- What is the effect of widespread deployment of DSS? Do clients still experience reduced response time?
- What is the effect of DSS on traffic volume? How is the distribution of traffic affected?
- Is the cost of network probing to implement DSS justified by the resulting performance improvement?

The first part of the paper presents a summary of our measurement techniques and motivation for the DSS approach. We then present the live Internet experiments that suggest the potential user benefits of DSS.

The second part of the paper presents the simulation study. We begin with a description of the simulation environment: the simulation engine, the network model and the traffic generation method. We then discuss the questions studied through simulation and present the results of the simulation experiments. We find that even with widespread use, DSS continues to benefit clients by reducing response time; while the overall data traffic volume is reduced and traffic distribution is improved as traffic is shifted from the backbone to regional and local networks.

2. Related work

2.1. Dynamic server selection

Dynamic server selection (DSS) was first described in [9]. In a DSS scheme, the assumed environment consists of replicated information servers, any one of which can satisfy client requests, as illustrated in Fig. 1. The client has no knowledge of the placement of the servers or the topology of the network between it and the servers.

In a static server selection scheme, the client is assigned to a particular server, most likely based on some static measure of distance such as network hops. However, it may be that using a server that is farther away can improve the client's response time. This is often true with globally distributed resources where servers and network resources half-way around the world may be less busy than local ones. Given this set of candidate servers, clients use assessment of network conditions to choose the server expected to provide the quickest response. Extensions to include assessment of server load and other information quality measures can be added to the DSS framework.

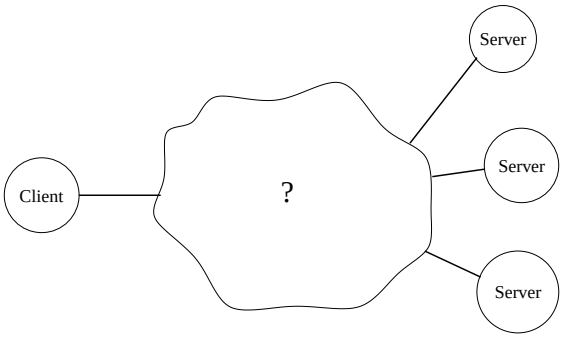

Fig. 1. Server selection model: the client can choose one of the servers.

Our work on server selection using end-to-end performance metrics is complementary to related work in the area. For example, the Harvest hierarchical cache resolution protocol [10] involves sending a message to all parent and sibling caches and using the one whose response is received first. In this paper, we show that the additional time required for more complete measurement of current network conditions often results in improved performance.

Application-layer anycasting has also been used in the context of server selection [4]. In that work an architecture is proposed in which an anycast resolver intercepts the anycast message and, based on cached information about available servers, can reply with a list of one or more alternatives. The selection criteria are very flexible and encompass both performance measurements (hops, throughput, etc.) and boolean policy predicates. This work is complementary to our work on measurement methodology and may provide an elegant solution to the problem of providing a list of alternatives that the client can use as input to DSS. Follow-on work [15] suggests the use of a resolver that maintains information about the best server for each client, based on both network and server conditions. That approach is primarily concerned with client-oriented metrics and is evaluated using an actual implementation in the Internet. In contrast the work described in this paper focuses on network metrics, and studies via simulation how the network specifically is affected by the use of server selection.

Server discovery and selection was also the focus of Guyton and Schwartz in [17], where the objective is to discover "nearby" servers, thus keeping

communication local and limiting the use of long-haul links. That work attempts to determine a subset of the network topology and build a distance metric on that topology. The metric used is hops and a method similar to triangulation is used to determine the distance to servers. This results in a static selection policy that does not consider the highly-variable nature of the network characteristics. In fact, the authors consider the high variance of round trip times a drawback. Conversely, 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 (RTT), 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. In contrast, our DSS policies 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.

In [18], a replication technique (and presumed server assignment) based on geographical distance is proposed. In their design, 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. Clients request the address of the nearest replica from the home server which calculates distances in miles to find the replica closest to the requesting client. This is essentially a static method of server selection and relies on the server maintaining a large amount of geographical information. In contrast, we show below that a DSS policy can provide high performance and a simplified placement policy.

Selection algorithms for replicated web servers are examined in [31]. Algorithms based on latencies of recent HTTP transfers are used to predict future transfer times; the server with the minimum recent latency is chosen. Comparisons to policies based on network hops or round-trip times show improvements of a factor of 2, reinforcing the results we present here. However, the authors do not consider the effects of multiple clients simultaneously employing these selection policies nor do they report on the network effects of their selection algorithms.

2.2. Dynamic path characterization

In order to minimize the time between document request and document arrival in a DSS scheme, the client needs to evaluate the quality of the connection to each of the candidate servers. In particular, a measure of network utilization can be used 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 have developed tools to measure the current utilization of a connection. These measurements are necessarily done from a distance and in an unknown environment. Therefore, we refer to our measurement process as *probing* and the tools as *probes*. 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. We summarize the essential features of the tools next. Full details and validation tests for both BPROBE and CPROBE can be found in [7,8].

2.2.1. BPROBE: *measuring base bandwidth*

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 typically 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. 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. In effect, BPROBE measures the bottleneck link of the round-trip path from the client to the server and back to the client as illustrated in Fig. 2.

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 interpacket 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 Fig. 3 (adapted from Van Jacobson [20]). In addition, Bolot describes the basic effect in [5,6] and Keshav has used a similar method in networks of Rate-Allocating servers [23,24].

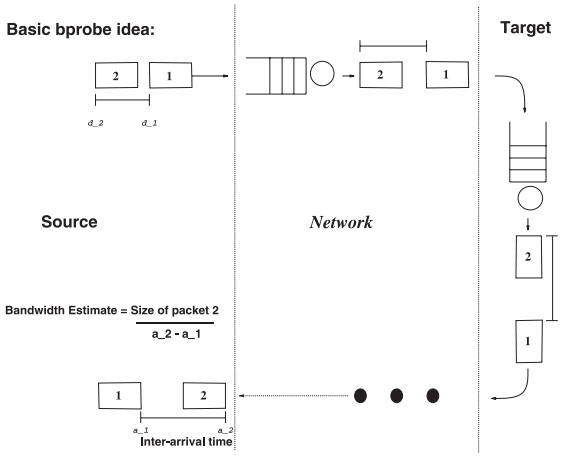

Fig. 2. Flow of packets from site of probe (client) to target.

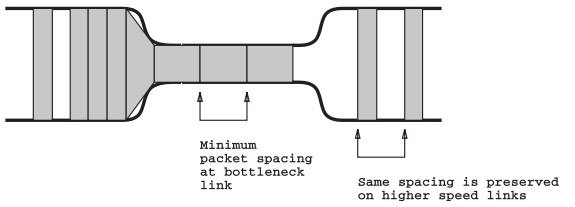

Fig. 3. Packet flow through a bottleneck link.

Pathchar [21,27] is a tool that does hop-by-hop characterization of network paths. The measurement mechanism is similar to ours: successive sets of probe packets are used to infer characteristics. While we concentrate on path characteristics, pathchar works on a hop-by-hop basis. By computing the difference between measurements at node $n+1$ and node n, the metrics (latency, bandwidth and queuing delay) for the link between the two nodes can be inferred. In order to generate this hop-by-hop detail for a network path, pathchar does require more network traffic and measurement time than our tools do. This is necessary because of the statistical nature of the techniques used to calculate link metrics based on differences between measured values. Such a level of detail is more than is needed for our purposes. Rather, our aim was to produce a robust, lightweight tool that measures characteristics of the path as a whole and is suitable for use by

applications needing a fast estimate of network conditions.

The aim of the BPROBE tool is to create this bandwidth-proportional packet spacing 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 $\mathscr{P}$, and the inter-arrival time (or *gap*), we can estimate the base bandwidth of the bottleneck link, $\mathscr{B}_{\text{bls}}$, as follows:

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

However, in contrast to the network architecture assumed in [23], in our experimental environment (the current Internet) this ideal behavior is not easily achieved. There are several problems that arise in practice: probe packets may not queue at the bottleneck link; competing traffic along the path may intervene between probe packets; packets sent by the probe may be dropped; and, 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 Fig. 3 and design an accurate, robust and low-impact tool to measure bandwidth.

There are two principal techniques with which we attack these problems. First, the use of multiple packets of varying sizes; second, a careful filtering process which discards erroneous measurements. In order to ensure queuing at the bottleneck router, we send many packets and we use packets of varying sizes. Larger packets will naturally take more processing time at routers and increase the possibility of queuing. 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. By sending a large number of packets, we also increase the likelihood that some pairs will not be disrupted by competing packets and we minimize the effect of occasional packet drops. 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. When queuing occurs on the return trip, after passing 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 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 nine inter-arrival measurements. Thus, at most seven sets of at most nine 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, Fig. 4 shows the raw estimates for five 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 five line segments of the graph, all those for the second on the next higher five 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.

Our filtering method is based on computing an "error interval" around each estimate, and combining these intervals using a set union operation. The error interval can be expanded as necessary until a satisfactory estimate of the speed of the bottleneck link is determined. The *union* filtering method combines overlapping intervals using set union, and selects an interval only if enough sets contribute to it. One interval is produced as the final result and the midpoint of this interval is returned as the final estimate.

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. It is these properties of the data, which are evident in Fig. 4, combined with our non-linear filtering approach which results in a reliable, robust tool for measurement of bottleneck bandwidth. Validation studies (detailed in [7]) in which BPROBE was run over the Internet found the following. Measurements of bottleneck links to nearby (campus Ethernet) hosts were within 10% of the true value 97% of the time. Measurements of bottleneck links to wide-area hosts (T1) were within 20% of the true value 82% of the time.

2.2.2. CPROBE: *measuring 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

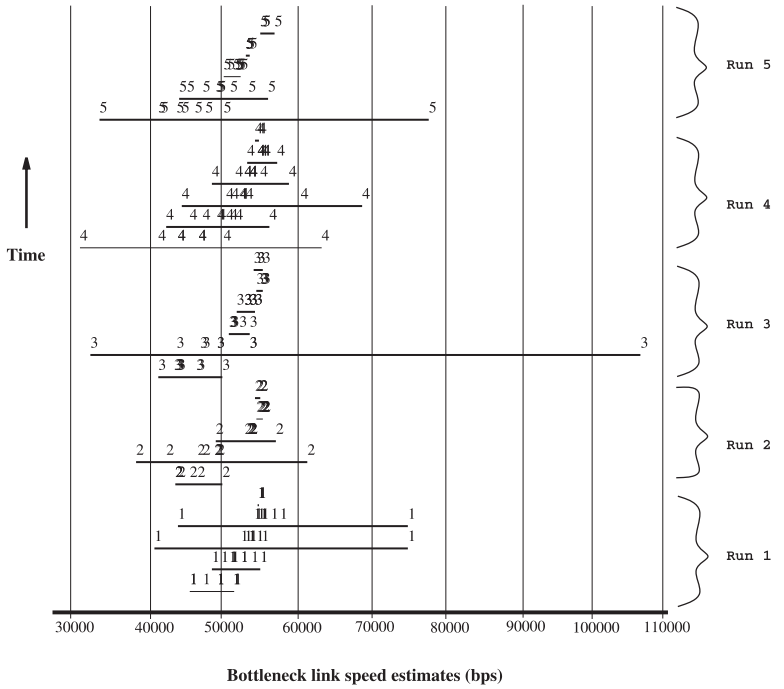

Fig. 4. Five BPROBE experiments to local 56 KB hosts.

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 the elapsed time yields a measure of available bandwidth, $\mathscr{B}_{\text{avail}}$, 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:

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

As in the case of BPROBE, care must be taken to ensure that the CPROBE's results are valid. Further discussion and validation studies may be found in [7,8].

Part I

Live experiments

In this first section, we describe our proposed solution to the server selection problem and evaluate it in terms of both user-oriented and system-oriented costs and benefits. This evaluation is done in the context of experiments based on empirical measurements of the operational Internet.

First, we divide the universe of documents into two classes: small and large documents. We expect the transfer time for small documents to be dominated by RTT, so we use simple, low overhead, RTT measurements as a basis for DSS. Then, for larger documents, where we expect bandwidth limitations will dominate the transfer time, we use a combination of RTT measurements and measurements of available bandwidth derived from our CPROBE tool. After considering the costs of various measurement approaches, we show that simpler measures can be as effective while costing less both in terms of measurement time and network bandwidth overhead. We then present our limited overhead DSS protocol suitable for documents of any size and show its improved performance compared to static policies and its minimal impact on network bandwidth.

Once again, our methods can be viewed from two standpoints: from a client-oriented standpoint, they reduce response time, which is good; from a system-oriented standpoint, they add traffic to the network, which is bad. In the first part of this section we concentrate on showing that response time can be radically reduced using (sometimes expensive) measurements. In the latter part of this section we show how almost all client-oriented benefits can be obtained using much less expensive, but approximate, measurements. Our final scheme obeys a strict limit of not increasing overall traffic in the network by more than 1%.

3. Dynamic server selection

3.1. Why dynamic server selection?

In any server selection scheme, a client seeking a document would like to choose the server which it has reason to expect 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 (RTT). However, the static nature of the hops metric (the number of hops between any two hosts rarely changes) is considerably different from the quickly changing, widely varying RTT metric. These differences are important in solving the server selection problem. We begin by comparing empirically measured distributions of each metric. We measured the values of

both metrics between a fixed client host and 5262 servers selected randomly from a list of WWW servers [19].

Fig. 5 shows in graph (a), the measured distribution of hops to the set of servers; and in graph (b), 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 (17 hops), whereas the distribution of latencies has a median (125 ms) much less than the mean (241 ms). The differences between the distributions suggest that hops is a poor predictor of latency.

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 Fig. 6, which presents measurements of latency to a single host gathered over a period of two days at intervals of 30 s. 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. A DSS policy can be designed to take advantage of exactly this sort of variation.

In fact, 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., [18]). 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

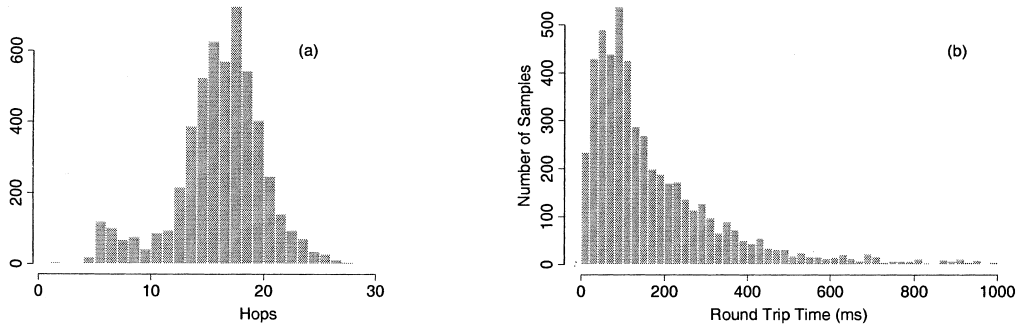

Fig. 5. Empirical distribution of hops and RTT to 5262 random servers.

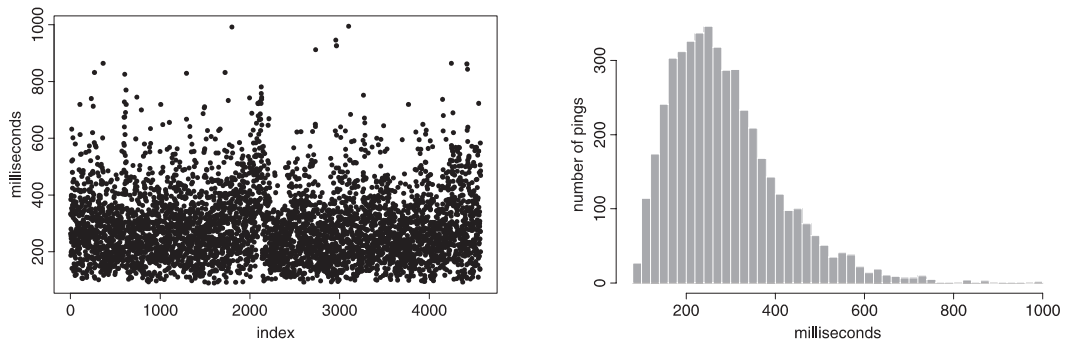

Fig. 6. Round trip times to a single host.

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 a replicated server system that uses DSS 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.

3.2. Server selection for small files – RTT

Having established the intuitive basis for dynamic server selection, in this section we present experimental results to show that even very simple DSS 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 [2,14]). 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 a system, we identified 10 hosts with approximately equal average round-trip latency as measured from our test client. Using *traceroute*, we measured the number of hops to each of the target hosts (we assume that this value did not change over the duration of our measurements). On each host we chose five files, one each of size 1, 5, 10 and 20 KB. We prepared a script which would periodically use *ping* to make five RTT measurements and then fetch each of the five documents and measure the transfer time for each one. Then, over a three-day period, for each host we measured (about once per hour) the round-trip latency using *ping*, and the transfer time for documents of sizes 1, 5, 10 and 20 KB. We chose these smaller file sizes as appropriate to test the simple RTT-based server selection methods since latency should be the dominant factor for small transfers.

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 mean of 1, 2, 3, 4 or 5 round-trip measurements. The simulation used a replication degree of seven (i.e., each client chose one of seven servers). For each trial, seven data points were chosen from the collected database and a choice was made for each of the eight polices. The transfer time for the chosen server was recorded as the result for the trial. The average transfer time over 1000 trials for each policy are summarized in Fig. 7. For comparison, we also include Optimal and Pessimal polices which are simply the minimum and maximum transfer times observed, respectively. [2] The graph shows the average time to fetch an object for each of the four file sizes. For the policies based on RTT measurements, the time plotted includes the time to make the required number of RTT measurements. Static policies are presumed to have no additional cost since the preferred server can be found by consulting a table.

All of these policies were evaluated with the objective of minimizing user response time. In this respect, as can be seen in Fig. 7, our experimental data reveal that the measurement-based 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 5 KB. In this simulation, the dynamic policy based on the mean of four RTT measurements gives the best results, minimizing the transfer time (inclusive of measurement time). Dynamic policies based on fewer (1, 2 or 3) RTT measurements do a little worse. A policy using five RTTs does worse still, exhibiting a phenomenon of diminishing returns for extra measurement overhead.

[2] Since we are simulating the algorithms off-line from measured data, for each set of seven chosen measurement we can simply choose the smallest transfer time as the Optimal and the largest transfer time as the Pessimal.

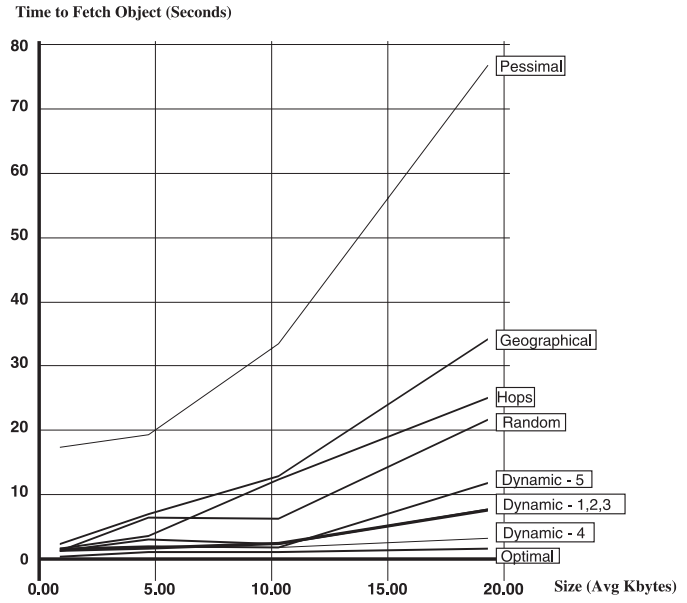

Fig. 7. Fetch times of static and dynamic policies plotted against document sizes.

When we compare the various selection policies to our hypothetical optimal transfer time we find that the best dynamic policy in this simulation required an average transfer time that was only four times the optimal for 5 KB files and this improved to less than two times optimal for larger files. In contrast, the static hops selection policy which was also about four times optimal for small files, showed increasingly worse performance as transfer size increased, up to 20 times optimal for 20 KB files. These results use seven replicas, certainly not a very large number for WWW document replication. The results would likely improve with a greater degree of replication.

However, the benefit of multiple measurements of RTT as a predictor of transfer time relative to the optimal value decreases as file size increases. This is clear from the increasing gap between predictions based on five RTT measurements and the optimal (minimum) transfer time shown in Fig. 7. We believe that this indicates the impact of limited available bandwidth due to competing traffic. Reducing the available bandwidth has the effect of increasing transfer time. This is especially true for larger document transfers while small document transfers have their transfer time dominated by instantaneous latency. This observation led us to formulate the previously mentioned techniques for measuring the bottleneck link speed and available bandwidth. Server load may also contribute to increased transfer times, but as we discuss further below, lack of a simple way of measuring server load prevented us from studying this directly. We next explore DSS using measurements of available bandwidth in addition to latency and its application to larger service times.

3.3. Server selection for large files – PTT

The results of the previous section clearly show the benefit of dynamic policies. However, the relative benefit was observed to decrease with increasing transfer size, a portion of this effect was hypothesized to be due to bandwidth limitations. In this section we present a server selection policy, called predicted transfer time (PTT). This policy, based on dynamic bandwidth measurements, is appropriate for larger files. We use the term *available bandwidth* to refer to the estimated transfer rate available to an application – that is, the portion of raw bandwidth not currently used by competing traffic.

We suspected that one of the factors involved in the reduction in predictive ability of the RTT policies as document size increased was limited available bandwidth. In order to estimate available bandwidth, we developed the CPROBE tool outlined above (and presented in detail in [7]). In this section we study the use of bandwidth measures concentrating on documents larger than those considered in Section 3.2. 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, two questions arise: (1) can we improve the predictive capability of our DSS algorithm? and (2) what is the additional cost of the probes?

3.3.1. Evaluating detailed server selection

In order to evaluate server selection policies based on available bandwidth measurements, we collected data from several WWW servers in the Internet as we did in Section 3.2. From earlier work [3] we have available a database of document size and location information. From this source we extracted sets of documents of sizes 100, 500, 750 KB and 1 MB. 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: five RTT measurements (using *ping*), the available bandwidth (using CPROBE) and the transfer time for the document.

Before beginning simulation of the various selection policies, 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 available bandwidth. Table 1 gives the R^2 values of the regressions. 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 a valid prediction window. As we have shown above, the effect of competing 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.

Table 1
Regression analysis: R^2 values for two metrics for four document sizes

Regression formula	Document sizes			
	100 KB	500 KB	750 KB	1 MB
xfer time vs. Single *ping*	0.053	0.186	0.477	0.081
xfer time vs. *Ping* and CPROBE	0.291	0.280	0.638	0.145

To use *ping* and CPROBE together, we formulate the predicted transfer time metric. 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:

$$\text{Predicted transfer time} = k_1 \text{RTT} + k_2 \frac{\text{document size}}{\mathscr{B}_{\text{avail}}},$$

where $\mathscr{B}_{\text{avail}}$ is the available bandwidth as measured by CPROBE. In the simulation of this selection policy PTT, the predicted transfer time, is calculated for all candidate servers and the server with the minimum value is chosen.

We again simulated both static and DSS algorithms using the data collected from our survey of WWW servers. For each document size (100, 500, 750 KB and 1 MB) we selected seven data points uniformly from the recorded data and applied the dynamic algorithm based on RTT and available bandwidth to make a selection from among the seven sites. The results are presented in Table 2. 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

Table 2
Simulation results for large transfers – mean transfer time in seconds

Selection policy	Transfer sizes			
	100 KB	500 KB	750 KB	1 MB
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
PTT	2.712	2.001	5.721	31.897
Optimal	0.985	1.296	4.265	17.529

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 seven servers; (2) Random, which picks a server uniformly; (3) Hops, which chooses the server which is the fewest hops away; (4) PTT, as defined above; and (5) Optimal, which chooses the best (smallest) transfer time.

The superiority of the dynamic policy for server selection is clear from Table 2. The improvement is especially marked for large documents. For example, for 500 KB documents, an improvement of over 75% is found when using PTT rather than Hops (2 s versus nearly 9 s). The transfer time chosen by the dynamic policy is less than two 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.

While the performance of PTT is good for larger files, there is some discrepancy in performance between 750 KB and 1 MB. One possibility is that the larger transfer takes long enough that the estimate of available bandwidth no longer applies. The arrival of new connections may mean that the available bandwidth will be less than originally measured. While we have found that our available bandwidth measurements are predictive for short intervals (e.g., the typical WWW request), certainly over the 10+ min for the worst-case 1 MB transfer time in our data sample, much can happen to invalidate the measurements. For instance, the arrival and departure of connections clearly affects the available bandwidth. A second possible explanation lies in our choice of hosts, a potential weakness in our study. Because it was difficult to find sites hosting 1 MB files we had fewer sites to choose from. It may be that there were some peculiarities about those sites or the paths to reach them that biased the measurements. This may also explain the fact that the Hops selection policy is so much better than Random selection in the case of 1 MB files, whereas in our other experiments, Random is usually a pretty good policy. If Random chooses a server that is much farther away and traffic is heavy it will surely suffer compared to Hops.

3.3.2. Impact of server load

As a validation step in our data analysis we studied the correlation between measured transfer rate and available bandwidth using the data collected in the PTT experiments described in the previous section. 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 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 are known to be popular such as *wuarchive.wustl.edu*, *www.ncsa.uiuc.edu* and *sunsite.unc.edu*. This suggests that server load should also be an input to a DSS algorithm.

As a first attempt to measure server load we tried the following technique: fetch a non-existent document from the target WWW server and measure the time required. The amount of time needed to fetch this empty page could serve as a proxy measure of server load. For this experiment we gathered data as in the PTT experiments in the

previous section with the additional preliminary step of fetching a nonexistent page and recording the transfer time. In general, this procedure results in retrieving a small (less than 200 byte) "error: URL not found" document. In a few cases, larger, personalized replies were received. We then ran another off-line simulation based on the measured data. There is some correlation between the larger test fetch times and serious overestimates 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 server selection based on this policy due to the high overhead of this method of server load measurement. In fact, our simulations show no improvement over the PTT policy. In practice, the cost of fetching a test document outweighs any benefit in improved prediction of transfer rate for the actual document. What is needed is a lightweight server load measurement method which can quickly and accurately assess server load. Perhaps the server itself should maintain a state variable that is easily queried. A tool that made such a query could be integrated with the tools we present here to significantly improve the server selection process.

4. Cost considerations

In addition to time overhead for network probing, which is reflected in the response time measurements we present, the impact on the network is also a critical issue. Probes that add many extra bytes of network load certainly will be unwelcome.

The current implementation of CPROBE uses four sets of 10 packets each with a maximum timeout of 1 s per set. Packet sizes of 600, 700, 800 and 900 bytes are used for a total overhead of 30 000 bytes and up to 4 s of measurement time. So, the probe may add up to an additional 30 000 bytes to the network load and a few extra seconds for an accurate measurement of available bandwidth. Such costs are too high for widespread use.

To address this, we explored measurement methods that can have much lower network impact and yet would approximate the more expensive CPROBE. In the next section we present such methods, and then propose a policy based on these lower-cost measurements which limits probe overhead to no more than 1% additional traffic in the network.

4.1. Approximate approaches to PTT

Previously we showed that, in theory, the bandwidth measurements combined with latency (RTT) measurements have greater predictive capability than latency alone. However, we suspected that a less expensive policy based on the mean of more than one RTT measurement, might provide a rough approximation to a bandwidth estimation since long RTTs may be indicative of low available bandwidth. First, we compare the performance of the combined policy (PTT) with the simpler RTT-only policies. Then we address the issue of cost of the additional measurements.

For these experiments we used data collected as in Section 3.3. From a set of servers, each approximately 100 ms away, we collected five RTT and data transfer time measurements. We performed this test once an hour over a three day period. We then performed off-line simulations of various selection policies based on our measured data. In Table 3 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 five *ping* measurements and chooses the server with the minimum value. Both of these polices were also used in the initial study described in Section 3.2. Once again, this table shows that for our data a measurement-based dynamic policy is always superior to the static policy based on distance measured using hops, confirming our earlier results. For large files, the use of either multiple RTT measurements or bandwidth probing improves over the results based on only a single latency measurement. For the 1 MB case the static Hops policy is superior to Dyn 1 but worse than Dyn 5 or PTT. This suggests the value of multiple RTT samples.

The surprising result is that the Dyn 5 policy which relies solely on RTT measurements gives results very close to those of PTT which uses the

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

Selection policy	Transfer sizes			
	100 KB	500 KB	750 KB	1 MB
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
PTT	2.712	2.001	5.721	31.897

Table 4
Correlation between available bandwidth measurement and RTT measurement (average of 1, 2, 3, 4 or 5 RTT measurements)

Number of pings				
1	2	3	4	5
0.236	0.293	0.346	0.372	0.400

additional measurement of available bandwidth. A similar though weaker correlation persists when using fewer than five RTT measurements. Table 4 gives values for the correlation coefficient between the reciprocal of the available bandwidth (measured using CPROBE) and the average of n RTT measurements for $n = 1, 2, 3, 4, 5$. Clearly the correlation increases with the number of RTT measurements, implying a strong link between the number of RTT measurements made and the degree to which those measurements capture the available bandwidth. As shown in the table, more RTT measurements give an estimate with greater correlation to the available bandwidth.

To summarize, both Tables 3 and 4 suggest that the policy of minimizing the mean of five *ping* measurements already accounts in a way for congestion effects. In other words, 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. These observations suggest a limited overhead DSS policy introduced in the next section.

4.2. The OnePercent protocol

Since the performance of the lighter-weight probes closely approximates the more expensive ones, we devised the following probing protocol. Paying particular attention to impact on the network, we propose a protocol that permits overhead that is typically not more than 1% of the request object size in terms of additional bytes injected into the network. We therefore refer to this policy as OnePercent. That is, for each document request, the number of additional bytes used to probe the network is normally at most 1% of the size of the document requested. [3] This protocol also ensures that the additional time needed to probe the network state is proportional to the document size. Thus, larger documents, which will benefit from more precise measurements of network conditions, are provided with better measurements while smaller documents for which latency is the dominant factor will not increase response time or use network bandwidth unnecessarily.

The OnePercent policy is formulated as follows: for files under 10 000 bytes, we use a single ping; under 20 000 bytes, we use the mean of two pings, etc. Because the CPROBE overhead is 30 000 bytes, all of the larger documents will use the mean of five pings in the OnePercent protocol. Thus, we use at most one ping per 10 KB to be transferred. Assuming a 100 byte ping packet we can meet the 1% threshold. Since this policy depends on the size of the object to be transferred it is only intended for those situations in which the size can be estimated (e.g., proxies reloading caches, estimates based on extension (.html vs. .gif vs. .mpg)). In practice, only a rough estimate of the size is

[3] For unusually small documents the overhead may be higher, but we would not recommend network probing for such small documents. For the experiments reported here the overhead was always under 1%.

necessary since even small amounts of probe information are useful in server selection.

Using the data collected in Section 3.3 we performed off-line simulations to understand how a selection policy with limited overhead would perform. Fig. 8 presents the results of applying the OnePercent protocol. For each document size, there are five bars showing the response (transfer) time in seconds for the best and worst transfer times (Optimal and Pessimal, respectively); for Random selection of server; for static selection based on minimizing the number of Hops; and for the OnePercent dynamic probing policy. In order to make visible the values for smaller documents, the bar chart was cut off at 100 s (the 1 MB Pessimal time is 700 s, the 1 MB Random time is 281 s). From the figure, it is clear that the OnePercent policy is in every case no worse than the static and random policies and in most cases performs close to Optimal. It is important to recall that the time for performing the measurement probes is included in the OnePercent values presented in the table. Thus, even with the measurement overhead accounted for, the OnePercent DSS policy is superior to static policies.

Turning to network load we find that in fact substantially less than 1% overhead was necessary to achieve these results. In fact, using the 500 byte (5100 byte ping packets) overhead for the larger files results in an aggregate network bandwidth overhead of about 0.1%.

4.3. Summary

Using empirical data collected from the operational Internet, we have shown that a measurement-based, dynamic approach to server selection can outperform a static selection scheme. In particular, user response time is decreased when clients choose a server based on recent measurements of network congestion. Further, we have shown that even simple surrogate measures of congestion (such as RTT) can provide much of the benefit of more precise measures for far less cost.

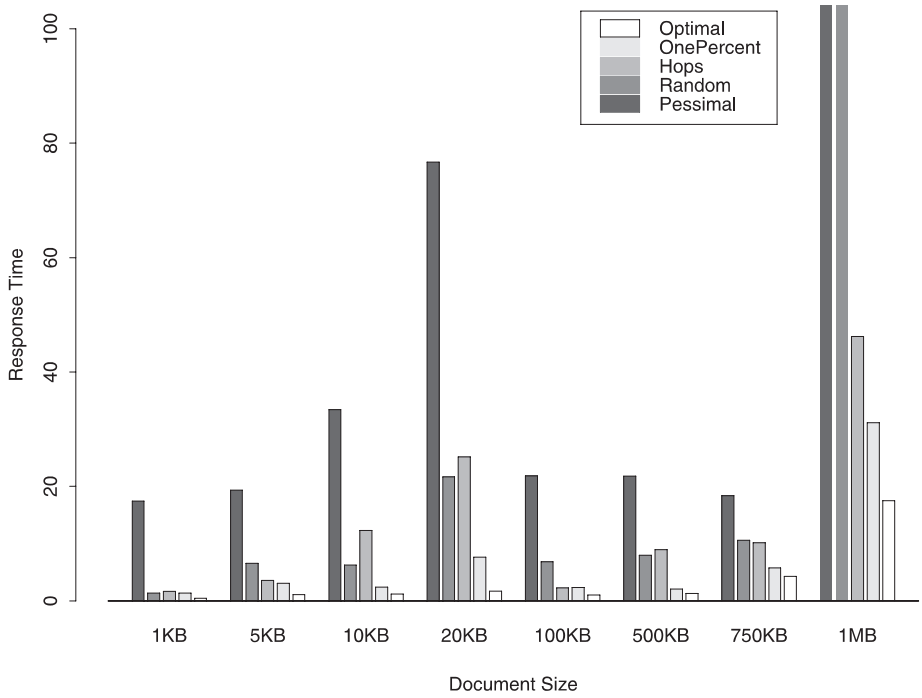

Fig. 8. Performance of OnePercent dynamic server selection policy.

Part II

Simulation experiments

The previous experiments demonstrated the benefit of a measurement-based, dynamic solution to the server selection problem. However, in those experiments, only a single client was using a DSS approach. What would happen if all clients used DSS? Although we were concerned with measurement overhead in terms of extra probe traffic, in our Internet experiments we could not measure the effect of DSS on network traffic volume or traffic distribution patterns. Using a simulated network we can explore questions about the system impact of DSS as well as examine implication of its widespread deployment. In this section we examine these questions:

- Suppose all client use DSS simultaneously. Do all clients still benefit?
- What are the effects of widespread use of DSS on network traffic volume and distribution?
- Does DSS tend to make requests travel farther in the network? If so, what effect does this have on the network?

Such questions are difficult to answer through experiments on the "live" Internet where so much of the network is beyond our control and difficult to measure. Through simulation we can control the experimental environment, measure much more carefully and run experiments to determine the system-level network effect of DSS.

5. Simulation engine

We used the `ns` network simulator [26] from Lawrence Berkeley National Laboratory (LBL). The primary reason was that `ns` faithfully reproduces the dynamics of TCP. This is important for any study in which the majority of traffic is controlled using the TCP protocol. Our study is based on HTTP transfers (which relies heavily on TCP for individual data transfers); thus a desirable characteristic of the simulation is realistic TCP modeling. The `ns` simulator maintains a faithful model of the TCP (Tahoe or Reno) state machine and implements the slow-start and retransmit behavior of these popular implementations. The realistic model of TCP behavior along with the network design described below increases our confidence that our results will transfer to real-world deployment and use.

The `ns` simulator is an event-driven network simulator implemented in C++ and Tcl. Nodes and links are the basic components from which networks are built. Nodes act as containers for agents which are sources and sinks for data.

For the purposes of this work it was necessary to make a number enhancements to `ns`.

1. In order to model the BPROBE and CPROBE tools, ECHO packets were added as a new packet type.
2. Client and server agents were defined as subclasses of the TCPAgent and TCPSink classes, respectively. The clients model the probing and selection of servers as well as data transfer. The servers act as sources of bytes for the clients and cooperate to maintain server wide statistics, such as load and amount of data requested.
3. A simple extension allows the measurement of the number of hops between nodes, necessary for the static selection policy based on hops.

6. Simulation model

6.1. Network topology

An important issue in any simulation study is the validation and verification of the simulation model. Zegura et al. [32], discuss the importance of matching the simulated network to the real-world network under investigation. Their findings are that a hierarchical model with average node degree of three reflect important attributes of the operational Internet. Our model closely follows these suggestions, with the simplification that the network graph is a regular one in order to facilitate aggregation of statistics.

Fig. 9 shows the network used in this work. We have chosen a three-level hierarchy with the highest level representing the backbone of the network; the intermediate level intended to represent

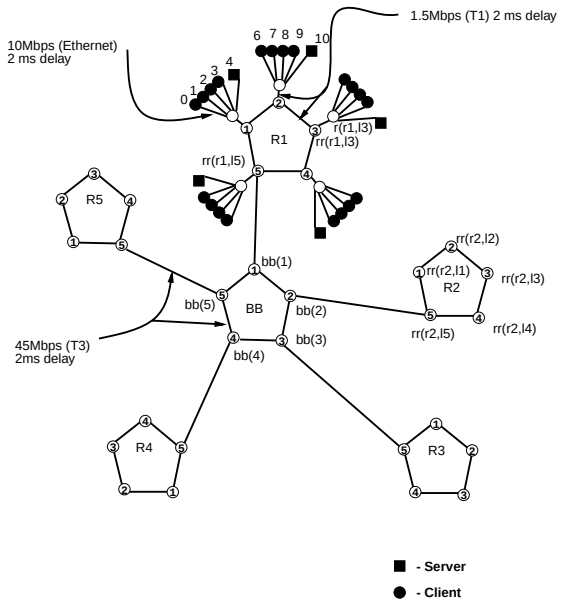

Fig. 9. Hierarchical network used in simulation experiments.

regional networks and the lowest level representing campus or enterprise LANs.

Referring to Fig. 9, the central pentagon and its adjacent edges represents the highest level of the hierarchy and models the backbone (BB) of the network. The BB nodes are connected in a pentagonal ring. The bidirectional links between nodes model T3 pipes with 45 Mbps capacity and 2 ms delay. Each of the five backbone nodes is connected via another T3 link to a ring of five nodes representing a regional network.

The regional pentagonal rings represent the regional networks (or ISPs) and the bidirectional links here model T1 (1.5 Mbps) links. To form the next layer of the hierarchy, each of the regional pentagons is connected via a T1 link to a LAN of five hosts.

The LAN level represents the campus or enterprise level network and is modeled by an Ethernet style network of five nodes. On each LAN, one of the five hosts is designated as a server machine (the squares) and the remaining four nodes (filled circles) are designated as clients.

In general the network can be described by a tuple of the form:

$$\langle NumRegions,\ LansPerRegion,\ HostsPerLan \rangle.$$

In the example shown in Fig. 9 we have a $\langle 5, 5, 5 \rangle$ network. Overall this network is composed of 180 nodes. There are a total of $NumRegions \times LansPerRegion \times HostsPerLan = 125$ hosts. Of these, 100 are designated as clients and the remaining 25 are designated as servers. There are 185 bi-directional links connecting the nodes and a total of 15 500 possible paths between leaf nodes (clients and servers).

6.2. Traffic generation

6.2.1. Traffic distribution modeling

Much recent work in the field of modeling traffic in computer networks [11,25] has suggested that Poisson traffic models do not capture adequately the high variability of network traffic. Specifically, it has been found that a characteristic of network traffic that is not captured by previous models is the heavy-tailed nature of the distribution of on and off times for data transfers. On times refer to periods during which messages are being sent or received and off times to periods when no network activity is occurring at a host. The heavy tail means that arbitrarily long on and off periods occur with small (but non-negligible) probability. This distributional characteristic, in turn, leads to long-range correlations between the density of traffic at widely spaced time intervals and can be seen in time series plots as burstiness at all time scales. In light of this research we chose to use a Pareto model for traffic generation.

The Pareto model captures the heavy-tailed nature of the distribution and is defined by its p.d.f.: $f(x) = \alpha k^{\alpha} x^{-(\alpha+1)}$ [22]. It is characterized by parameters k and α. The shape parameter, α, determines the weight of the tail, with smaller values resulting in a heavier tail. The parameter k is a scaling factor for the distribution (see Section 7.4 for more details). Pareto variates for the simulation are generated using the inverse transformation method. A uniform variate, u, is chosen from the uniform distribution $u \sim \mathrm{U}(0,1)$ and the Pareto variate is generated by

$$p = \frac{k}{u^{1/\alpha}}.$$

Below we study the sensitivity of our results to the precise value of α but for reasons of simulation efficiency we primarily report results for $\alpha = 1.5$.

The use of the Pareto distribution as we have done is supported by empirical studies of network traffic and user behavior. We use Pareto generation for the sizes of the data objects moved since it has been observed that files available on the WWW tend to have such a distribution with a power-law tail [11]. The Pareto distribution is also used for user inter-request times or user "think times". Such a heavy-tailed distribution of inter-request times has been observed empirically in data collected over long periods [25,13].

6.2.2. Traffic generation per client

The basic component of the simulation consists of a client and the stream of requests that it generates. Each client is modeled as a process which repeatedly performs a sequence of four actions:

1. (Optionally) Probe the network for performance information.
2. Select a server from a set of candidate servers (possibly chosen based on information gathered in Step 1).
3. Request a data object from a server.
4. After transfer completes, remain idle for a period of time.

6.3. Simulation parameters

The combination of simulation parameters was chosen to create a simulation recreating realistic network conditions to the extent feasible. We wanted relatively high link utilization in order to test the candidate selection algorithms under difficult conditions. Link bandwidths, data packet size and the α and means of the Pareto distributions helped ensure this.

6.3.1. Running time

The first parameter of interest is the running time. In order to make fair and repeatable comparisons between policies, each of the 25 clients executing the server selection algorithms makes the same number of selection decisions and data requests (300) in each simulation run. The remaining 75 clients generate repeatable background traffic as explained later. The running time was then adjusted such that each of the 25 active clients was allowed to make their 300 requests before the simulation terminates.

6.3.2. Link utilization

The early experiments were run on the network parameterized as denoted in Fig. 9. However, with the relatively small number of clients, utilization of links was only around 1%. With such unrealistically low network utilization, there is no clear necessity for network measurements, since all paths are lightly loaded. We ran a few test with more clients in order to load the network. This produced the more realistic traffic loads and patterns we expected but, due to the packet-by-packet simulation technique, this also resulted in extremely long simulation runs. In order to simulate the high traffic conditions expected in real networks in a reasonable amount of simulation time we introduced high utilization of the network links (90%) by scaling the capacity of the backbone links by a factor of 0.001. Thus, the effective capacity of the backbone links was 0.05 Mbps.

6.3.3. Pareto parameters

The size of the object requested as well as the idle time are chosen from a heavy-tailed (Pareto) distribution as described above. For all simulations described in Part II of this paper, the mean file size was 1000 bytes and the mean inter-request time at the clients was 0.001 s.

The value of α is 1.5 for all simulation results reported in this paper unless otherwise noted. This is a compromise between simulation realism and practical concerns. It is now believed that real-world systems exhibit a high degree of heavy tails (that is, small values of α) [1,11] but it is also known that simulations at small values of α exhibit instability and must be run for extremely long durations before stability is achieved [16]. For this reason we chose to use an $\alpha = 1.5$ for most of our simulations. This let us model traffic behavior realistically yet within practical time bounds.

6.3.4. Selection policies

Each client can behave in one of three different modes. In order to compare competing polices in a

fair way, the majority (75 of 100) of the clients generate a repeatable background traffic load which consists of repeating the last three steps of the basic client algorithm (choose a server uniformly from the set of servers, fetch a data object and rest for a time). This mode of operation (identified as RAND in the discussion of alternative selection policies below) is intended to model the current situation in the WWW in which there is no choice among replicated sources of information and a user in a typical browsing session essentially walks randomly over the graph of web servers.

The remaining 25 clients make up the set of clients of interest in this study: those that select servers by various policies. They can either operate in this same basic mode (RAND) as the majority of the clients or employ a more intelligent server selection algorithm. The first alternative is to use the probing step to determine the distance in hops to the candidate servers and then choose the server which is the smallest number of hops away from the client (this is referred to as the HOPS policy below). The next alternative uses active probing for available bandwidth and selection based on the server to which there is currently the greatest amount of bandwidth available (this is referred to as the DSS policy in the sequel).

6.3.5. Number of replicas

For all experimental results presented here we fixed the number of replicas at seven. That is, the client selects from an set of seven candidate servers. At each selection decision this set of seven servers is selected randomly from the 25 available servers.

Increasing the number of replicas should benefit both the DSS and HOPS policies. On the other hand, fewer replicas may provide most of the advantage of choice. We do not study the effect of varying the degree of replication here.

6.3.6. Pseudo-random variate generation

Each client model uses three independent streams of pseudo-random numbers when making choices as it proceeds. One stream is used in the process of selection of candidate servers to be probed (Step 1); a second stream is used for determination of the size of the requested object (Step 2); and the third stream is used to decide the length of the idle period (Step 3). The independence of the streams is guaranteed though the use of the `nrand48` C-library call which can be passed a previous value from which to generate the next random variate. In a pre-processing step, 1000 seeds were chosen at intervals 100 000 samples apart from a single stream of pseudo-random numbers. These "entry points" can then be used to isolate independent sub-sequences which are the streams used, three at a time, for each client. The same seeds are used to initialize each simulation run so that the clients doing selection request the same sequence of files (in terms of size and inter-request times) over all of the polices compared.

7. Metrics

There are several yardsticks that are used in the measurement of the simulated network. On the client side we focus on user-perceived latency. In terms of network performance we use two metrics: overall network traffic (measured in byte-hops); and traffic distribution over portions of the network hierarchy. In the remainder of this section we motivate and explain each of these metrics in turn.

7.1. User-perceived latency measurements

For this study we consider all replicas to be of equal quality in terms of information recency, authority, etc. The replicas are mirrors in the sense that they are interchangeable, identical copies. Therefore, the client's only objective is to minimize waiting time for each transfer. We measure user-perceived latency as the time elapsed between the moment a client requests a document and the moment when the information has been completely delivered to the client.

In the following sections, latency data are generally presented as a distribution over document sizes. This presentation preserves any effects due to bandwidth limitations on transfer times. That is, variation in transfer time due to document size is visible in these plots and not hidden as would be the case if a single aggregate throughput measure were given. There is an important distinction when

predicting transfer time between small and large files. For small files, latency is the dominant factor, while for large files, available bandwidth is more important.

The simulation runs with a fixed-size packet of 512 bytes for data transfers. This size, while perhaps smaller than typical for the Internet, allows us to more easily study the interactions between packets, by making them more frequent, while running simulations in a reasonable amount of time. Given 512-byte packets, we then compute histograms with 512-byte bins when presenting the user latency data. However, the nature of the underlying file size distribution requires some more careful aggregation as explained in Section 7.4.

7.2. Aggregate byte-hops

A measure of network load is the amount of data flowing through the network. This directly affects router queue lengths, packet drops, packet retransmissions and packet delivery times – all measures of interest to network providers and managers. In terms of the current work, it is important to establish: at least (1) that the proposed algorithm does not adversely affect the network by imposing an undue amount of additional load on the network; or preferably (2) that the proposed algorithm alleviates network overloading.

We measure load in *byte-hops* which counts each byte once for each link it travels over. This captures the total amount of data being sent but scales this number by the "distance" over which it travels. While it is desirable to reduce the volume of bytes sent, which can be accomplished, for example, through caching at the client, it is also important to reduce the distance over which bytes are sent and to account for retransmissions at their full cost. The byte-hops metric lets us compare in a fair way the difference between a few-hop, heavily-loaded path (which may suffer from reduced throughput capability and even dropped packets) with a many-hop lightly-loaded path (which may often be the better choice resulting in faster transfer time and, since there are no drops and corresponding retransmissions, fewer byte-hops).

The experimentally observed network load reported below includes only the traffic involving the 25 monitored hosts. Since we are comparing the amount of network load induced by a fixed set of requests across various selection algorithms we only want to aggregate the traffic across those hosts. Because the simulations run for a fixed set of requests and not a fixed amount of time, the volume of background traffic will differ from one simulation to another. Therefore we do not want to aggregate all requests but rather we need to isolate the traffic generated by the fixed set of requests made by the 25 monitored hosts.

7.3. Distribution of aggregate traffic over network hierarchy

A desirable goal of policies that affect network traffic is to keep traffic local when possible. That is, when given the chance, it is preferred to use resources that can be found at the LAN or regional network level rather than those that are located across the backbone. While [17] focuses exclusively on this measure as the goal of a host selection protocol, attempting to keep traffic off of long-haul links, we find that this effect is achieved as a by-product of our DSS protocol. Our experimental results indicate that DSS tends to choose local servers more often than remote ones.

To quantify this effect we again measure byte-hops and plot the fraction of total aggregate traffic that passes over each link in a particular portion of the network. As discussed in Section 6.1 our test network is hierarchical and consists of backbone, regional and LAN tiers. Each link in the network is assigned to one of these three tiers and the fraction of the total network traffic that flows over each link is computed. Then, for each tier, the average fraction is computed over all links in the tier. In this way, traffic distribution patterns for the candidate selection algorithms may be compared.

7.4. Non-uniform histogram bins

Due to the heavy-tailed nature of the Pareto distribution, a non-uniform bin size is needed in presenting the distributional data in the following sections. The file sizes are generated using a Pareto distribution and are interpreted to be numbers of

bytes, but it is appropriate to use bins that are multiples of the fixed-size 512-byte packets when reporting data relative to file size. Since the file sizes are *not* distributed uniformly over the range of sizes it was necessary to introduce a non-uniform width bin scheme. The motivation behind the technique was to attempt to define bins such that all would contain approximately the same number of observations. Since the bulk of the distribution is biased toward small documents, we want the resulting histogram to have smaller bins for smaller documents with bin size increasing with document size.

The overall effect of the non-uniform bin technique is to smooth the data presented, since there are roughly equal numbers of observations in each bin and averages over these bins then have meaning. Still, it is the case that the tail of the distribution is sparsely populated. In practice, this means that measurements towards the tail of the distribution are less reliable since fewer observation are made in that range.

To begin, let us examine the underlying distribution from which file sizes and user think times are generated. The CDF for the Pareto distribution with scaling factor k and shape parameter α is given by [22]:

$$P[X \leqslant x] = 1 - \left(\frac{k}{x}\right)^{\alpha} \tag{1}$$

with mean

$$\frac{\alpha k}{\alpha - 1}, \quad \text{where } \alpha > 1,$$

and variance

$$\frac{\alpha k^2}{(\alpha - 1)^2(\alpha - 2)}, \quad \text{where } \alpha > 2.$$

Based on the value of α, bins were constructed as follows. For $x < 10\,000$, that is, file sizes less than 10 000 bytes, equal sized bins (integer multiples of the (512 bytes) packet size used in the simulation) were used. Thus, the first 19 bins have right-hand edges at $(512, 1024, 1536, \ldots, 9728)$ these capture files from 1 through 19 packets in length. Above this size we wish to define bins that will be approximately equally populated. This is achieved by dividing the remaining probability mass into 15 equal-sized pieces and computing the right-hand bin edges for each of these bins.

The remaining portion of probability mass outside the 19 fixed-size bins can be computed from Eq. (1) as

$$P[X > x] = 1 - P[X \leqslant x] \tag{2}$$

$$= 1 - \left(1 - \left(\frac{k}{x}\right)^{\alpha}\right) \tag{3}$$

$$= \left(\frac{k}{x}\right)^{\alpha}, \tag{4}$$

which can be divided into equally-size pieces to get an increment value: *ProbIncr*.

Eq. (1) can be solved for x, the value below which a desired fraction lies:

$$x = \frac{k}{(1 - P[X \leqslant x])^{1/\alpha}}. \tag{5}$$

The remaining bins can be computed from Eq. (5) given a value for $P[X \leqslant x]$, starting with the value given for $[P \leqslant 10\,000]$, and incrementing by *ProbIncr* on each iteration. For example, for $\alpha = 1.5$ the extra bins (rounded to multiples of the packet size) have right-hand edges of: (10 752, 11 264, 11 776, 12 800, 13 824, 14 336, 15 360, 16 896, 18 432, 20 992, 24 576, 29 696, 38 400, 60 928). All remaining observations (those larger than 60 928 bytes) are discarded because the wide variety of values in this last bin renders aggregate statistics meaningless. There is just too much disparity for an average user-latency, for example, to make sense at this range.

8. Results

This section presents the results of our wide-area internetwork simulation answering the questions we raised at the outset.

8.1. Impact of DSS on user-perceived response time

The first question we addressed through simulation is: what is the effect of large-scale deployment of a dynamic server selection policy? Some potential problems that could arise are: overload-

ing of network resources due to probe traffic; "hot-spots" in the network created when many requests are directed to the same server(s); and overloading of servers. We also want to confirm that, from the client point of view, the benefit of DSS still applies even when all the client are using this approach.

In order to examine these questions we ran three sets of simulations, one for each candidate server selection policy. With 75% of the clients generating background traffic we set up the remaining 25% of the clients to use either the RAND, HOPS or DSS selection policies. For each set we ran four experiments using different seeds for the random number generator and averaged the results.

In Fig. 10, the user-perceived latency is plotted against the data object size for the three tested selection policies (RAND, HOPS and DSS). The *y*-axis shows the average elapsed time between request and the delivery of the last byte for data transfers (the user-perceived latency). The *x*-axis shows the data object size in bytes. It can be seen from Fig. 10 that the DSS policy improves user response time over the RAND policy by approximately a factor of 2 for small files and by about 0.75 for larger files. The improvement is even stronger when compared to the HOPS policy (the top line in the figure). This echoes the earlier results for the Internet experiments: static polices based on HOPS do worse than a random selection among candidate servers.

8.2. Effect of DSS on data traffic volume

The question of total data traffic is also important: will the widespread use of a DSS policy increase data traffic or might it lessen it? In Fig. 11 the number of byte-hops required to transfer all the data for the 25 test clients are shown for the three candidate selection polices. Recall that the total amount of data bytes requested is the same for the three cases. Thus the totals shown in the graph reflect the data traffic required to deliver a fixed workload. There are several possible explanations for the differences between policies: (1) since each hop adds to the total, packets that

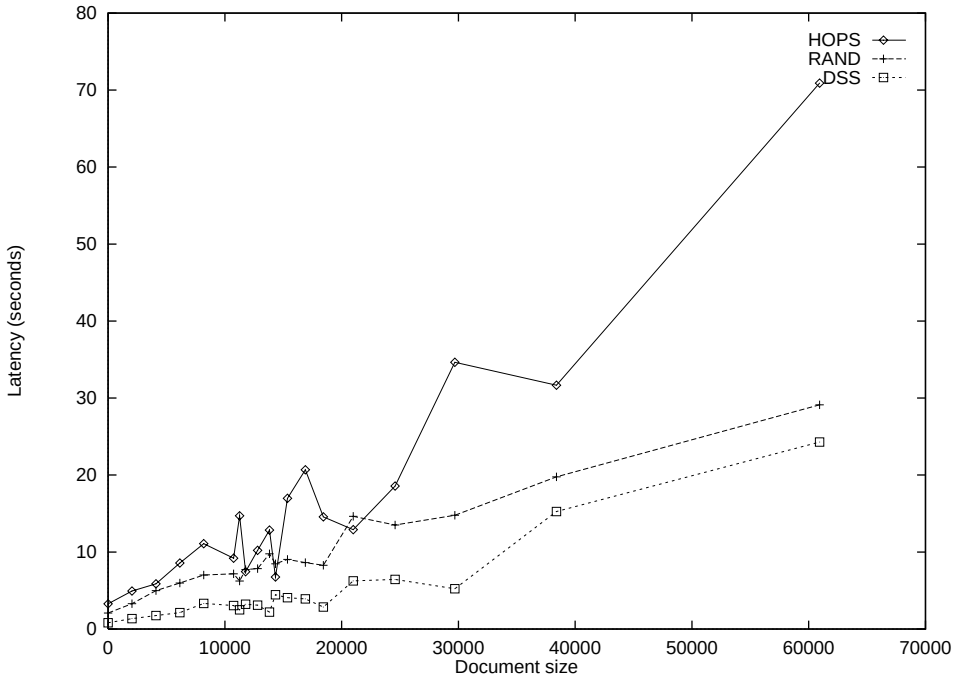

Fig. 10. User-perceived latency: comparison of three selection policies.

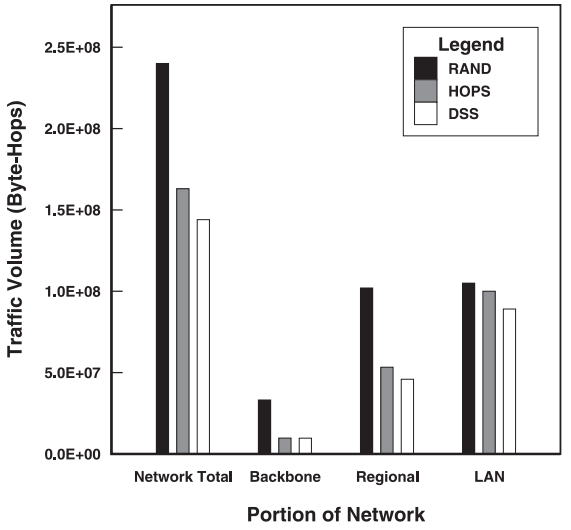

Fig. 11. Traffic volume (byte-hops): comparison of three selection policies.

travel on longer paths add more than shorter paths; (2) retransmissions caused by congestion losses on heavily-used paths also add extra traffic. We examine the contributions of both of these factors below.

On the left of the graph are three bars giving the total amount of traffic network-wide (that is, the sum of bytes over all links for the simulation). The three polices are shown from left to right, in decreasing order, with the RAND policy requiring the most traffic and the DSS policy generating the least. The HOPS policy performs well, needing only 67% of the byte-hops used by RAND. The DSS policy uses only 60% of the RAND byte-hops. With seven replicas, the DSS policy reduces traffic overhead by 40% over the RAND policy.

To the right of the figure, we break down the total traffic by hierarchical tier. The majority of the savings are at the backbone and regional tier, where both HOPS and DSS require less than half of the byte-hops needed for the RAND selection policy (about 30% each for the backbone and around 50% for the regional tier). On the far right, for the LAN tier, the results are closer, with the DSS policy at 90% of RAND and HOPS at 95%. This begins to show the effect of favoring local servers (avoiding the heavily congested backbone links), a common outcome for both the HOPS and DSS selection policies.

To more completely understand the mechanisms that result in this overall traffic reduction, let us look at the two components introduced above: distance measured in hops and retransmissions resulting from packet loss. Table 5 shows the measurements of data traffic volume, average distance and packet loss for the RAND and DSS selection policies. [4] The results shown here are aggregated over all the 25 clients utilizing server selection and represent the bytes transferred from the chosen servers.

8.2.1. Distance in hops

We can compute the average distance (measured in hops) for each selection policy by dividing the total traffic volume in byte-hops (column 2 of Table 5) by the amount of data requested (column 1 of Table 5). We find that the RAND policy fetches data from an average distance of 10.7 hops (which is nearly the maximum path length of the simulated network). The DSS policy results in a much shorter average distance of 6.4 hops. Thus, part of the explanation for reduced traffic volume is the shorter distances over which data travels in the DSS case. In other words, the DSS policy results in similar benefits to a policy based on minimizing the distance measured in hops.

8.2.2. Retransmission behavior

The other factor contributing to reduced traffic is the reduction in the amount of retransmitted data. In order to measure this, we instrumented the simulator to measure the number of bytes retransmitted by the servers in response to packet losses. The values for each selection policy are shown in column four of Table 5. We can convert this to a packet loss percentage (since packets are of fixed size) by dividing by the volume of data requested. We find that the RAND policy results in a packet loss rate of 4.3%, which is close to recently reported values for live Internet measurements [28]. However, clients using DSS expe-

[4] Data shown in Table 5 were taken from a shorter simulation run than shown in Fig. 11.

Table 5
Data traffic volume breakdown: average distance and packet loss

Selection policy	Data volume (MB)	Traffic volume (MB-hops)	Average hops	Retransmissions (KB)	Pkt loss (%)
RAND	15.5	167.3	10.7	673	4.3
DSS	15.5	98.9	6.4	418	2.7

rience a packet loss rate of only 2.7%. Since the servers chosen by DSS are on paths that are less heavily loaded, the chances of a time-consuming packet loss, which results in increased user latency as well as increased traffic volume, appears to be reduced significantly. Thus the use of DSS appears to result in a lower packet loss rate which explains part of the reduction in data traffic volume.

8.3. Effect of selection policy on distribution of traffic

A further demonstration of the improved distribution of traffic observed in the previous section can be seen in Fig. 12 where the percentage of byte hops per tier is shown. This was computed by summing the number of byte-hops over all links in a tier and dividing by the total number of data byte-hops for the simulation. The RAND policy spreads equal amounts of traffic (43%) on both the regional and LAN tiers and a smaller proportion (12%) on the backbone. The HOPS and DSS policies, on the other hand, have the smallest proportion of traffic on the backbone (about 6%) and increasing amounts on the regional (about 30%) and LAN (about 60%) tiers.

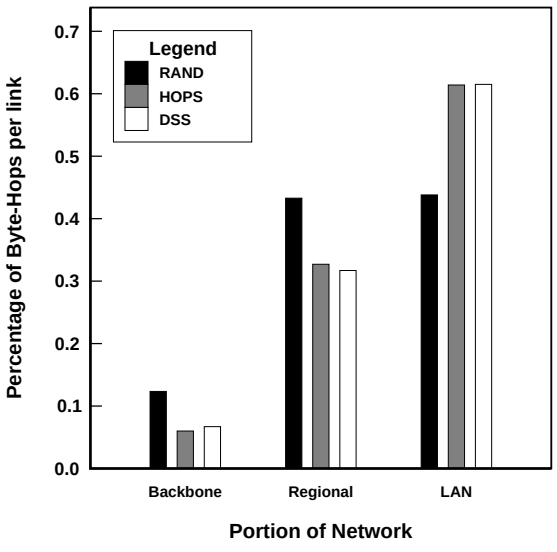

Fig. 12. Distribution of traffic by network portion: comparison of three selection policies.

It is not surprising that the HOPS policy tends to have more local traffic since its objective is to minimize the number of hops over which the data travel. However, it is interesting to note the few cases in which the DSS policy travels further in the network (as seen from the slightly greater fraction of backbone traffic and slightly smaller fraction of regional traffic compared to HOPS) and yet requires fewer total byte-hops (Fig. 11) and less time on average (Fig. 10).

8.4. Reducing DSS overhead through approximation

In Section 4.2, we introduced the OnePercent dynamic selection protocol. In that scheme, a client uses far fewer probe packets than the DSS policy we have examined so far. With OnePercent, an additional network load of at most 1% of the bytes of the document being transferred is used to probe the network. The form of these probe packets is, again, ICMP ECHO (*ping*) packets and the comparison metric is the mean of the RTTs of these packets. So, instead of a large overhead of packets delivered by the network and additional overhead of calculation to estimate in a robust way the available bandwidth of the path, OnePercent uses the mean of a few packet RTTs and chooses the path with the smallest mean RTT. As we suggested previously, if this policy can achieve similar performance results, it is preferable because of its reduced overhead.

We simulated the OnePercent policy and compared the results with the more expensive

CPROBE-based DSS policy. For these simulations, we used $\alpha = 1.5$ and a mean file size of 3000 bytes. Each policy processed 18 750 transfers. Fig. 13 shows the results of these comparisons. Here we show the user-perceived latency (in seconds) plotted against document size for the DSS and OnePercent dynamic selection policies. The dark line (with '+' markers) shows the results for the DSS policy, while the gray, line (with '◇' markers) gives the results for OnePercent.

Fig. 13 shows that the two dynamic selection polices have very similar performance curves. The user latencies for the two policies are very close, interleaving results over the entire range of document sizes.

Due to the small size of the files used in our simulations, the average number of ECHO packets implied by the 1% overhead of the OnePercent policy is a single packet. This is sharply contrasted to the 30 packets used by the current CPROBE implementation. Clearly, approximately equivalent results for much less cost is a significant advantage of the simpler policy. As also stated earlier, there may still be an argument for the use of the more expensive probe policy for larger document sizes. The graph in Fig. 13 is inconclusive on this point although there may be a trend toward divergent performance as file sizes get very large.

A comparison of volume of network traffic for the two dynamic selection policies is presented in Fig. 14. Once again, the two are essentially identical in terms of total byte-hops for the data traffic. The OnePercent policy also shows a similar reduction in the packet loss rate and a similar average distance as the DSS policy. Either DSS policy results in nearly the same amount of traffic generated by transferring the selected files.

8.5. Sensitivity to α

All of the traffic streams used in the simulations are generated using Pareto distributions as described previously. The α parameter of the Pareto

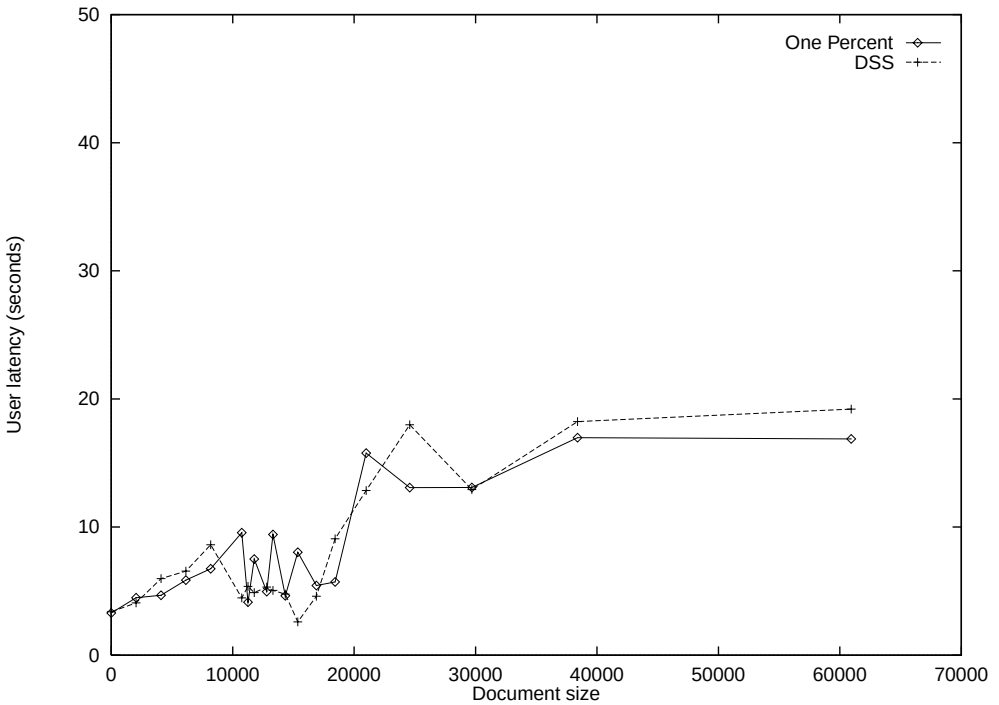

Fig. 13. User-perceived latency: comparison of CPROBE and OnePercent selection policies.

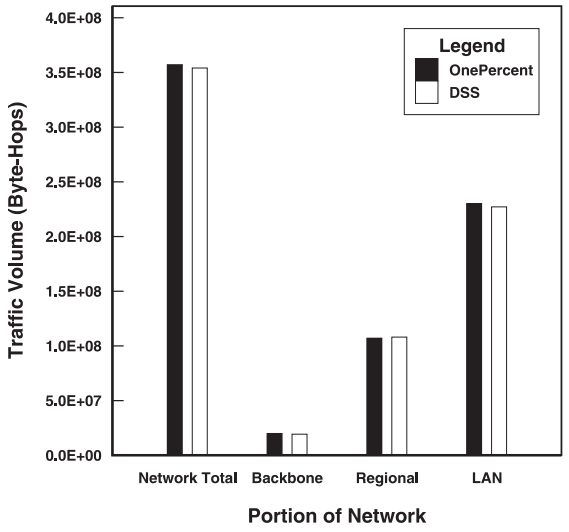

Fig. 14. Network traffic: comparison of CPROBE and OnePercent selection policies.

distribution determines the shape of the curve, that is, the weight of the tail, with values closer to 1 being more heavy-tailed and values closer to 2 less heavy-tailed. Simulations using small α values must be run far longer than those that use larger values of α in order to achieve stability [16]. However, there has been much work in the area of characterization of current network loads which suggests that realistic network traffic is quite heavy-tailed. This implies that simulations should use values of α close to 1 to duplicate real-world results. But, with current simulation approaches this often results in impractically large running times.

In spite of this practical limitation, we want to find an indication as to how well the proposed selection algorithms will perform under different workload characteristics. That is, we want to compare the performance of the RAND selection and the DSS selection policies while varying the weight of the tail of the Pareto distribution. We chose to run pairs of experiments (RAND and DSS) with five values of α: $(1.1, 1.3, 1.5, 1.7, 1.9)$.

In order to make faithful comparisons we need to ensure that the mean request size is equivalent across all the experiments. Recall that the mean of the Pareto distribution depends on the shape parameter, α, and the scaling parameter, k, and is given by

$$\text{mean} = \frac{k\alpha}{\alpha - 1}.$$

We chose to keep the mean request size fixed at 1000 bytes and then computed the corresponding value of k for each α used in the experiments.

Fig. 15 is a plot showing the performance of the RAND selection algorithm relative to the DSS algorithm as α is varied. Each line in the graph plots the ratio of user-perceived latency for RAND selection to user-perceived latency for DSS selection. The vertical axis gives the ratio between the two selection algorithms, and the horizontal axis give document size in bins of 512 bytes. Values plotted are interpreted as the factor by which the RAND selection under-performs the DSS selection (measured by increased response time) for a given value of α and a given document size. A horizontal line is also plotted at a factor of 1. Above this line, the DSS policy outperforms the RAND policy. Below the line the converse is true. For instance, a value of 2 on the vertical axis means that a user choosing a server based on the RAND policy would wait twice as long as a user choosing a server based on the DSS policy. Except for the case of ($\alpha = 1.7$, document size $= 1024$) all points show that DSS performs better than RAND. While there is some indication that DSS performs better with more heavy-tailed distributions it is not clear that there is any significant trend as α is varied and we cannot say that DSS is very sensitive to α.

The conclusion to be drawn from this data is that while DSS is in general a preferred method of server selection when compared to RAND selection, it appears especially useful when traffic is characterized by a heavy-tailed object size distribution such as has been found in empirical studies of current network and workloads as confirmed by many researchers [11,13,29,30,1]. More extensive measurements and simulations will be required before this point is well established. However, these preliminary results certainly bode well for the user of DSS in real world networks.

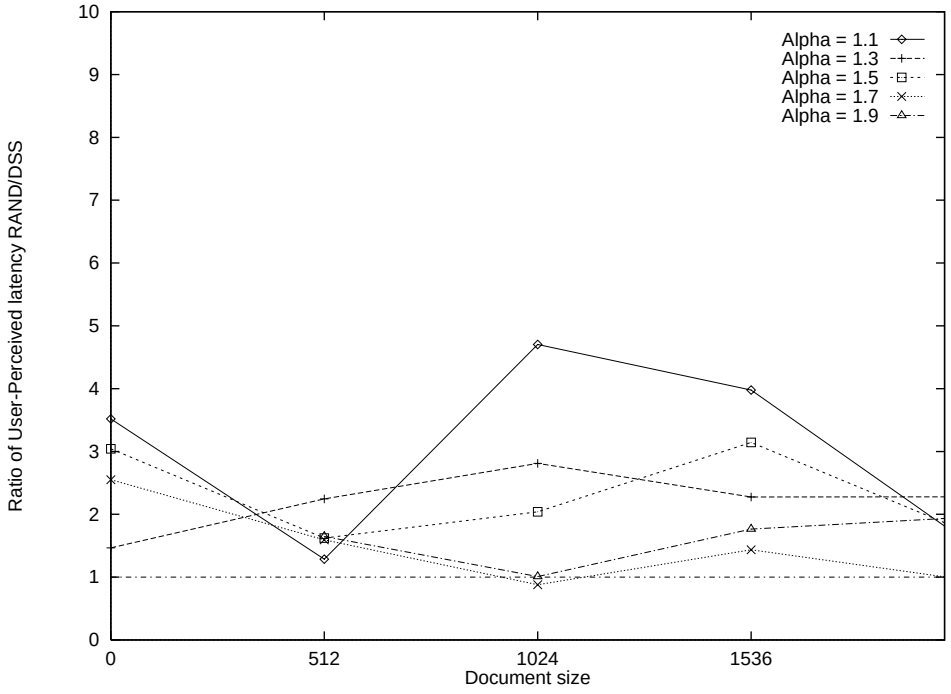

Fig. 15. User latency: ratio of RAND to DSS vs file size as α is varied.

9. Conclusion

Widespread replication of information can ameliorate the problem of server overloading but raises the allied question of server selection. Clients may be assigned to a replica in a static manner or they may choose among replicas based on client-initiated measurements. The latter technique, called dynamic server selection (DSS), can provide significantly improved response time of users when compared with static server assignment policies (for example, based on network distance in hops).

In the first part of this paper we have demonstrated the idea of DSS using experiments performed in the Internet. We described a range of policies for DSS and showed that obtaining additional information about servers and paths in the Internet before choosing a server improves response time significantly. We compared the effectiveness of policies that explicitly measure network bandwidth to simpler policies that only measure round-trip delay and showed that the latter approach generally provides nearly equivalent performance at much lower cost in terms of additional network traffic. The best policy we examined adopts a strategy of never adding more than 1% additional traffic to the network, and is still able to provide nearly all the benefits of the most expensive policies.

While these results suggest that DSS is beneficial from the network user's standpoint, the system-wide effects of DSS schemes should also be closely examined. Since previous work has looked at DSS in isolation, it is important to determine whether in widespread use DSS would continue to be beneficial. In addition, it is hard to predict how DSS would affect the global flow of traffic in a large internetwork.

In the second part of this paper we used large-scale simulation to study the system-wide network impact of DSS. We used a simulated network of over 100 hosts that allows local-area effects to be distinguished from wide-area effects within traffic patterns. We generated network requests in a manner intended to imitate Web traffic patterns,

and used a detailed packet-level simulation that includes TCP flow control.

In this environment we have compared DSS with static server selection schemes and confirmed that client benefits remain even when many use DSS simultaneously. Importantly, we have also shown that DSS confers system-wide benefits from the network standpoint, as compared to static server selection. First, overall data traffic volume in the network is reduced, since DSS automatically improves network congestion. Second, traffic distribution improves – traffic is shifted from the backbone to regional and local networks.

As the size of the Web and the Internet continue to explode, more sophisticated middleware to manage latency and improve throughput becomes more desirable. The results presented here are suggestive that widespread use of DSS would improve the perceived performance of distributed information systems like the Web, while at the same time improving Internet traffic distribution properties.

References

[1] M.F. Arlitt, C.L. Williamson, Web server workload characterization: the search for invariants, in: Proceedings of the 1996 SIGMETRICS Conference on Measurement and Modeling of Computer Systems, 1996, pp. 126–137.

[2] A. Bestavros, Demand-based resource allocation to reduce traffic and balance load in distributed information systems, in: Proceedings of the SPDP'95: The Seventh IEEE Symposium on Parallel and Distributed Processing, San Anotonio, TX, October 1995, pp. 338–345.

[3] A. Bestavros, R.L. Carter, M.E. Crovella, C.R. Cunha, A. Heddaya, S.A. Mirdad, Application-level document caching in the Internet, in: Proceedings of the IEEE SDNE'95: The Second International Workshop on Services in Distributed and Networked Environments, June 1995.

[4] S. Bhattacharjee, M.H. Ammar, E.W. Zegura, V. Shah, Z. Fei, Application-layer anycasting, in: Proceedings of the IEEE INFOCOM '97, Kobe, Japan, April 1997.

[5] J.-C. Bolot, Characterizing end-to-end packet delay and loss in the Internet, Journal of High Speed Networks 2 (3) (1993) 305–323.

[6] J.-C. Bolot, End-to-end packet delay and loss behavior in the Internet, in: Proceedings of the SIGCOMM 1993, ACM SIGCOMM, August 1993, pp. 289–298.

[7] R.L. Carter, M.E. Crovella, Measuring bottleneck link speed in packet-switched networks, Performance Evaluation (Proceedings of Performance '96) 27&28 (1996) 297–318.

[8] R.L. Carter, M.E. Crovella, Measuring bottleneck link speed in packet-switched networks, Technical Report BU-CS-96-006, Boston University, March 1996.

[9] R.L. Carter, M.E. Crovella, Server selection using dynamic path characterization in wide-area networks, in: Proceedings of the IEEE INFOCOM '97, Kobe, Japan, April 1997.

[10] A. Chankhunthod, P.B. Danzig, C. Neerdales, M.F. Schwartz, K.J. Worrell, A hierarchical internet object cache, in: Proceedings of the 1996 USENIX Technical Conference, San Diego, CA, January 1996.

[11] M. Crovella, A. Bestavros, Self-similarity in World Wide Web traffic: evidence and possible causes, in: Proceedings of the SIGMETRICS '96, 1996.

[12] C.R. Cunha, Trace analysis and its applications to performance enhancements of distributed systems, Ph.D. Thesis, Boston University, Boston, MA, 1997.

[13] S. Deng, Empirical model of WWW document arrivals at access link, in: Proceedings of the International Communications Conference – ICC'96, Dallas, TX, IEEE Press, New York, June 1996.

[14] Y. Endo, J. Gwertzman, M. Seltzer, C. Small, K.A. Smith, D. Tang, VINO: The 1994 fall harvest, Technical Report TR-34-94, Department of Computer Science, Harvard University, 1994.

[15] Z. Fei, S. Bhattacharjee, E.W. Zegura, M.H. Ammar, A novel server selection technique for improving the response time of a replicated service, in: Proceedings of the IEEE INFOCOM '98, San Francisco, CA, USA, 1998.

[16] M. Greiner, M. Jobmann, L. Lipsky, The importance of power-tail distributions for telecommunication traffic models, Technical report, Institut für Informatik, T.U. München, 25 August, 1995.

[17] J.D. Guyton, M.F. Schwartz, Locating nearby copies of replicated Internet servers, in: Proceedings of the SIGCOMM '95, Boston, MA, August 1995, pp. 288–298.

[18] J. Gwertzman, M. Seltzer, The case for geographical push-caching, in: Proceedings of the HotOS '94, 1994.

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

[20] V. Jacobson, Congestion avoidance and control, in: Proceedings of the SIGCOMM '88 Symposium on Communitations Architectures and Protocols, Stanford, CA, August 1988, pp. 314–329.

[21] V. Jacobson, Pathchar – a tool to infer characteristics of Internet paths, Lawrence Berkeley National Laboratory, alpha edition, April 1997, Available at: ftp://ftp.ee.lbl.gov/pathchar/msri-talk.ps.gz.

[22] R. Jain, The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling, Wiley, New York, 1991.

[23] S. Keshav, A control-theoretic approach to flow control, in: Proceedings of the SIGCOMM 1991, ACM SIGCOMM, pp. 3–15.

[24] S. Keshav, Packet-pair flow control, Available at http://netlib.att.com/netlib/att/cs/doc/94/2–17.ps.Z, 1995.

[25] W. Leland, M. Taqqu, W. Willinger, D. Wilson, On the self-similar nature of ethernet traffic, in: Proceedings of the SIGCOMM '93, September 1993, pp. 183,193.
[26] S. McCanne, S. Floyd, ns – LBNL Network Simulator, Lawrence Berkeley National Laboratory, Obtain via: http://www-nrg.ee.lbl.gov/ns/.
[27] B. Nitzan, E. Nemeth, Pathchar – man page (alpha version), http://www.nlanr.net/k/pathchar.html, 1997.
[28] V. Paxson, Measurements and analysis of end-to-end internet dynamics, Ph.D. Thesis, University of California, Berkeley, Lawrence Berkeley National Laboratory, April 1997, LBNL-40319; UCB//CSD-97-945.
[29] D. Peterson, R. Grossman, Power laws in large shop DASD I/O activity, in: CMG Proceedings, December 1995, pp. 822–833.
[30] D.L. Peterson, Data center I/O patterns and power laws, in: CMG Proceedings, December 1996.
[31] M. Sayal, Y. Breitbart, P. Scheuermann, R. Vingralek, Selection algorithms for replicated web servers, in: Workshop on Internet Server Performance (Sigmetrics '98), 1998.
[32] E.W. Zegura, K. Calvert, S. Bhattacharjee, How to model an internetwork, in: Proceedings of the IEEE INFOCOM '96, San Francisco, CA, USA, 1996.