Mitigating Server-Side Congestion in the Internet Through Pseudoserving

Keith Kong and Dipak Ghosal

***Abstract*— Server-side congestion arises when a large number of users wish to retrieve files from a server over a short period of time. Under such conditions, users are in a unique position to benefit enormously by sharing retrieved files. Pseudoserving, a new paradigm for Internet access, provides incentives for users to contribute to the speedy dissemination of server files through a contract set by a "superserver." Under this contract, the superserver grants a user a referral to where a copy of the requested file may be retrieved in exchange for the user's assurance to serve other users for a specified period of time. Simulations that consider only network congestion occurring near the server show that: 1) pseudoserving is effective because it self-scales to handle very high request rates; 2) pseudoserving is feasible because a user who participates as a pseudoserver benefits enormously in return for a relatively small contribution of the user's resources; 3) pseudoserving is robust under realistic user behavior because it can tolerate a large percentage of contract breaches; and 4) pseudoserving can exploit locality to reduce usage of network resources. Experiments performed on a local area network that account for the processing of additional layers of protocols and the finite processing and storage capacities of the server and the clients, corroborate the simulation results. They also demonstrate the benefits of exploiting network locality in reducing download times and network traffic while making referrals to a pseudoserver. Limitations of pseudoserving and potential solutions to them are also discussed in this paper.**

***Index Terms*— Caching, flash-crowd, Internet server technology, pseudoserving.**

I. INTRODUCTION

SCARCE bandwidth remains a problem: surveys continue to report long download times as the number one reason users are dissatisfied with the Internet [9]. To better meet the demand for bandwidth, the research community is responding with innovative technologies on several fronts. A first response is the development of faster network components, including modems, switches, and transmission lines. Despite these improvements, a flourishing user population and the introduction of new multimedia applications continue to demand even more bandwidth. Internet phone, videoconferencing, and downloads of large multimedia files, for example, remain tolerable only under the best of network conditions.

Manuscript received August 22, 1997; revised April 16, 1999; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor S. Pink. This work was supported by the National Science Foundation under Grant ANI-9714668.

K. Kong is with the Department of Electrical and Computer Engineering, University of California at Davis, Davis, CA 95616 (e-mail: kkong@ece.ucdavis.edu).

D. Ghosal is with the Department of Computer Science, University of California at Davis, Davis, CA 95616-8562 (e-mail: ghosal@cs.ucdavis.edu).

Publisher Item Identifier S 1063-6692(99)07037-5.

A second response is the development of new protocols that use bandwidth more efficiently. Examples of these include Compressed Serial Line Internet Protocol (CSLIP) [26] and low-bandwidth X [13], which use compression techniques to reduce redundancy. Other protocols such as Hypertext Transfer Protocol version 1.1 (HTTP 1.1) [11] and Transaction Transmission Control Protocol (TTCP) [27] improve on current standards by removing overhead. Still others call for the removal of entire protocol layers. Work is underway, for example, to deploy the Internet protocol (IP) directly over the Synchronous Optical Network (SONET).

A third response is multicasting. It works by aggregating a large number of requests and broadcasting the server's data to them at once. This avoids repeated usage of the same links to serve a large number of clients individually. Multicasting has been used successfully for a number of applications. Thus far, these applications have focused on the distribution of data, such as audio and video, where reliability is not an overriding concern. Research is currently underway to develop a reliable version of multicasting [22], [23].

A fourth response is the development of caching mechanisms within the Internet [4], [10], [21], [25]. These schemes work by recognizing that files are often requested more than once. By storing popular files locally, future requests for these files can be satisfied quickly without the need to retrieve them from the server. Caching schemes are characterized along a number of dimensions. Two of the most important ones include the location of the cache and the degree of cooperation [7], [14]. Data transfers from a cache that is close to the client tend to be faster and demand less resource from the network. Caches that cooperate tend to have fewer cache misses than ones that operate alone. Although caching schemes work well and are responsible for much of the reduction of bandwidth usage today [1], [2], [5], they do not always satisfy requests. This happens when the request is a first request or the requested data in the cache has become stale and needs to be refreshed by retrieving it from the server.

Closely related to caching is prefetching. Rather than keeping retrieved data locally on behalf of future requests, prefetching works by transferring data to the user *before* they are needed. Mailing lists, network news, and so-called "push" technologies belong to this category; data is pushed from the server to the client in anticipation of future requests. This technique has two main drawbacks. From the standpoint of the user, prefetching does not improve response time if the file requested has not been prefetched. From the standpoint of the user community, prefetching adds to congestion unnecessarily

when prefetched files are never requested. Aggressive use of prefetching has the potential to cause more delays than it saves.

A fifth response utilizes basic principles of economics [16]–[19], [24]. It recognizes that bandwidth is a scarce resource and seeks ways to allocate it optimally. Work in this area is often concerned with maximizing the welfare of the user community. This is usually done by granting priority for the delivery of packets to users who value it more at the expense of those who value it less. To encourage the truthful revelation of user values, these schemes often institute some form of pricing based on usage of bandwidth. Mackie–Mason and Varian's smart market [18] gives a flavor for how economics can be applied to the allocation of bandwidth. In it, packets are routed based on bids placed by users; packets with higher bids are routed with higher priority over ones with lower bids. While schemes like the smart market maximize the welfare of the user community, they tend to be impractical from a number of standpoints. Prioritizing packets based on bids, for example, requires that all routers cooperate. This requires significant changes to the well-entrenched IP protocol and is therefore difficult to implement. Moreover, pricing schemes based on usage often incur significant accounting costs, and basic questions such as who should be billed in a distributed connectionless environment such as the Internet are difficult to answer [16].

A. Pseudoserving: Caching + Economics

Pseudoserving [12] provides a new response to the bandwidth problem by combining elements of caching and economics. It hinges on the observation that users in possession of popular files from a busy server are in possession of a valuable resource. If they can be convinced to share this resource, congestion near the server can be avoided, and the welfare of the user community would benefit enormously.

Pseudoserving provides the necessary incentives for users to share files through a contract. In it, the server agrees to provide information on where the requested file may be immediately obtained. In exchange, the client agrees to "pay" by serving the retrieved file to other users within a short period of time. This payment is used to satisfy requests by other users. We show later that participants in this exchange can often reduce total retrieval times by *over an order of magnitude.*

Pseudoserving exhibits unique characteristics stemming from the interplay of cooperative caching with economics. Unlike caching schemes, there are no cache misses; the contract between the superserver and the pseudoserver ensures that resources are reserved to meet requests as they come. This contract also makes possible cooperation at an interorganizational level. In pseudoserving, users belonging to different organizations retrieve files from each other. In doing so, they promote sharing of cached data at a level greater than is currently done using caching schemes that operate only within organizations. Pseudoserving is also unique for being a testbed for a network bartering system. As in pricing schemes, users who pay receive better service. Unlike pricing schemes, however, this better service is not provided at the expense of other users. Because what users "pay" is precisely the goods in demand, resources are created as necessary as demand for them grows. In this respect, pseudoserving is *self-scaling.*

The remainder of this paper is organized as follows. Section II describes the various types of congestion and identify those types that can be eliminated or reduced by pseudoserving. Section III describes the architecture of a pseudoserving system. Section IV presents results from simulations illustrating the application of pseudoserving to dissipating flash-crowds. Section V describes an actual implementation of pseudoserving and discusses the results of running it on a local area network (LAN). Section VI discusses limitations of pseudoserving. Finally, Section VII concludes with a summary and a discussion of future work.

II. Types of Congestion

Broadly speaking, the bottlenecks experienced by data traversing through the links of the Internet can be categorized according to their location. We identify three types of bottlenecks: those that occur near the server, those that occur near the client, and those that occur at the intermediate links and nodes.

Server-side bottlenecks occur when a large number of clients are connected to the server at the same time. For many applications, such as the File Transfer Protocol (FTP), there is a limit to the number of connections that can be handled by the server simultaneously, called server concurrency [26]. Once this limit has been reached, no new connections can be established without additional ones being terminated. Because the ratio of price to performance of computer systems continues to drop, concurrency is becoming less of an issue, and server-side bottlenecks are moving toward the server's link to the Internet, where it is much more expensive to add capacity. When the server's link becomes the bottleneck before server concurrency is reached, the number of simultaneous transfers grows until the transfer rate to most clients is much less than the rate that their links to the Internet can handle. This is a growing problem for clients that access globally popular sites, where the competition for bandwidth is intensifying.

A similar problem occurs at the client side. Typically, a large number of clients share the same link to the Internet through an Internet Service Provider (ISP). The bandwidth of this link is divided among all of the clients when they are using it to transfer files simultaneously. However, client-side bottlenecks differ from server-side ones because users can avoid them. One can choose different levels of user sharing by subscribing to different services. A person who only uses the Internet for e-mail, for example, may not mind sharing her Internet connection with many other users. Users who demand faster connections can subscribe to a service where the number of users per link is smaller. In fact, users can have their own dedicated links if they are willing to pay for it.

Finally, bottlenecks occur at the intermediate links and nodes of the Internet. This happens during peak hours of network usage, during which many connections between servers and clients exist. Because packets belonging to many connections are handled by the same intermediate links and nodes, these links and nodes are often points of congestion. File

transfers that cross many such links often take an intolerably long time. As is the case with server-side bottlenecks, the user is powerless in avoiding bottlenecks arising from congested intermediate links.

Pseudoserving allows individual users to bypass congestion occurring near the server and at intermediate links and nodes of the Internet. This is made possible by the storage and bandwidth resources *users* can contribute to the network during periods of congestion. A *pseudoserving system* provides the mechanism to harness these resources. We describe it in the following section.

III. The Pseudoserving System

A pseudoserving system consists of two components: a *superserver* and a set of *pseudoservers.* The former grants the latter access to files in exchange for some amount of network and storage resources through a contract. This amount is zero under low-demand conditions, when the superserver functions as a concurrent server and pseudoservers function as clients. Under high-demand conditions, when the superserver's concurrency limit has been reached, it may be possible to grant immediate access to the pseudoserver in exchange for temporary usage of its network and storage resources. Under these circumstances and subject to the condition that the contract is met, the superserver gives the pseudoserver a referral to where the requested data may be obtained.

The following sections describe the components of a pseudoserving system in greater detail. For clarity, these descriptions are based on a specific implementation of such a system. For convenience, we refer to a pseudoserver before it has retrieved its requested file as a client. We distinguish such a client from the client of a traditional client/server system by referring to the latter as a *traditional client.*

A. The Pseudoserver

The pseudoserver plays essentially the same role in accessing data as that of the traditional client. However, there are a few key differences in the messages the pseudoserver sends to the server and the responses expected. These differences are noted below.

- A pseudoserver sends more information to the superserver in making a request than a traditional client does to a server. In addition to the name of the file requested, the client sends information about the resources it is willing to give in exchange for immediate access to the file. These resources include the time interval and number of clients within that interval for which it will act as a server.
- The response a pseudoserver expects from a superserver is also more varied than what a traditional client expects from a server. The superserver may send the file directly to the pseudoserver, in which case, the pseudoserver is asked to contribute some amount of resource not exceeding the ones the pseudoserver is willing to give at the outset. It is worth noting again that this amount may be zero, for example, under conditions of reduced demand. The client may be told that it is not possible for it to be served immediately. In this case, the client is also told the time it is expected to wait before access to the file is granted as a function of the resources it is willing to contribute. Finally, the pseudoserver may be given a referral to a host from which it can immediately retrieve the file if the contract has been met. As a measure of security, a cryptographic checksum for the file is sent along with the referral. Its purpose is to allow the pseudoserver to detect whether the file it has retrieved from the referred host has been modified.

At this point, it is worthwhile to briefly clarify what meeting a contract entails. In exchange for being served or given a referral, a pseudoserver guarantees to hold the file it just retrieved for some period of time. Within this *contractual file-holding time,* it agrees to serve the file up to some number of times as specified in the contract. If no clients arrive within this period, the pseudoserver is released from its obligation. It is also released from its obligation as soon as it has served the number of clients it agreed to serve, regardless of whether it has held the file for the agreed duration.

B. The Superserver

The superserver answers requests according to the flowchart shown in Fig. 1. This section explains the figure, making references to the numbered items in it. Let C denote the number of concurrencies the superserver has allocated to serving traditional clients and pseudoservers that have not met the contract, and let PSC denote the number of concurrencies the superserver has allocated to serving pseudoservers that have met the contract.

We begin with the top-most box. Requests are handled differently depending on the size of the file requested. If the file is small, the client is served immediately by the superserver (1). Otherwise, the superserver checks to see if a pseudoserver for the requested file exists (2). If one exists (3) and the contract conditions are met, the superserver tells the client the location of the nearest pseudoserver containing the file, along with the resources actually required from the client (4). Information regarding these resources and the IP address of the client are then stored in the superserver's main memory, indexed by the file name and the location of the client based on its IP address. This information is used for the benefit of future requests for the same file.

If no pseudoserver for the file exists (5), but the number of pseudoservers the superserver is concurrently serving has not reached PSC (6) and the contract has been met, the superserver itself serves the file to the client (7). If this limit has been reached (8), but the number of traditional clients the superserver is concurrently serving has not reached C, then the superserver serves the file to the client (9). This "freebie" route is also followed whenever a contract has not been met (10). This is to ensure that the superserver gives at least as much access to all hosts, regardless of their ability to contribute resources, as a traditional concurrent server would to a client. Finally, when it is not possible to give a referral to a client and it is not possible to serve a client because the server's concurrency for both have been reached, the client is told to retry later (11).

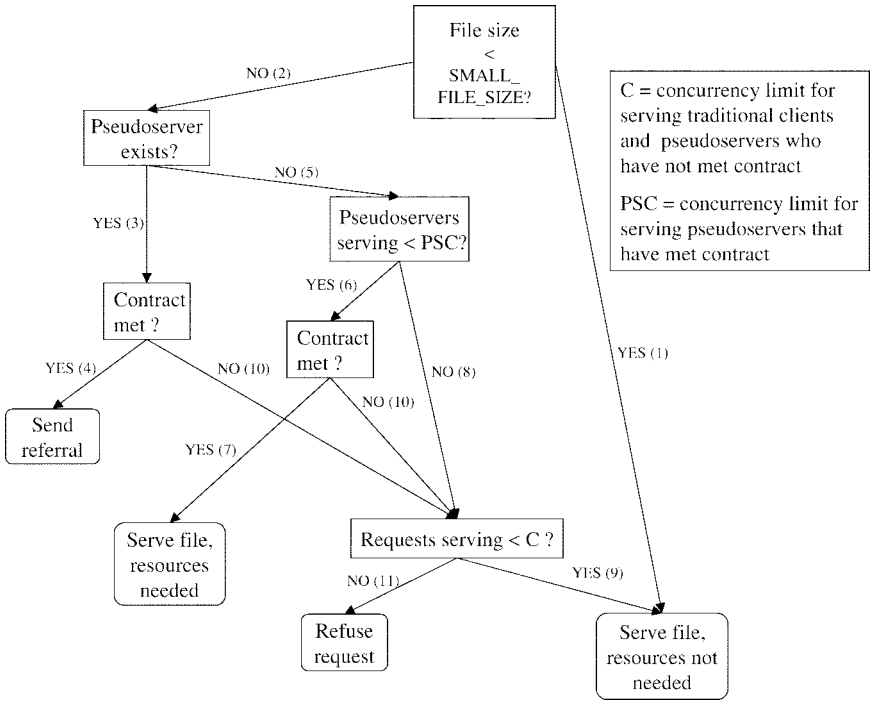

Fig. 1. Superserver flowchart.

C. Contract Policies

The key to a superserver's power is its ability to set contracts. In particular, it can set contracts according to demand for its resources. As demand increases, the superserver can set contracts that favor the creation of additional resources. As demand subsides, fewer resources are needed and thus the conditions of the contract can be relaxed. This section considers requirements that guide the establishment of effective contract policies and proposes such a policy.

In order to guarantee speedy access to users, two conditions need to be met. The first relates to the access of the requested file. In particular, a pseudoserver must exist that can satisfy a request as it arrives. This can be achieved by accumulating pseudoservers as necessary until the number of pseudoservers is sufficiently large to handle the rate at which requests arrive. This, in turn, can be achieved by stipulating in the contract that pseudoservers must handle more than one referral. If each pseudoserver handles N referrals before it leaves the *pseudoserver pool,* then with time, denoted by t, the number of pseudoservers in the pool grows as $N^{\mathrm{time/D}}$, where S is the time it takes a pseudoserver to serve N referrals. This *expansion mode* continues until the size of the pseudoserver pool is equal to the product of the request rate and the time it takes to download the file. When this happens a requester can be given a referral as soon as it arrives,[1] and the contract can be reduced so that each pseudoserver needs to serve only one referral to maintain the size of the pool.

Under such circumstances, the superserver is in the *steady-state mode.* If the rate of requests should drop, fewer pseudoservers are needed, and the superserver can enter the *contraction mode;* clients that make requests during periods of reduced demand are simply given referrals without any need for their resources. If there are as many periods of contraction as there are periods of expansion, a pseudoserver handles on average only one referral.

The second condition relates to the distribution of the requested file. In particular, the links and nodes between a requester and the pseudoserver from which it retrieves the file must be uncongested. To meet this requirement, it is necessary that clients can retrieve files from sites without traversing congested links. This requires knowledge of global network traffic and therefore can not be done easily without incurring a large amount of overhead. A more reasonable approach would be to relax the condition; rather than looking for a solution that guarantees files can be retrieved from sites that do not traverse congested links, we look for a solution that guarantees files can be retrieved from sites that are, on average, less congested than if the files were directly retrieved from the server. This can be done in the framework of pseudoserving by setting contracts based on the pattern of requests coming from groups of closely-linked networks, or *network clusters,* and making referrals only to a pseudoserver located in the same network cluster as the client. Section IV-B-4 discusses this in more detail.

IV. Dissipating Flash-Crowds

Flash-crowd conditions arise whenever a large number of requests are made over a short period of time for a small set of files contained on a server. This happens, for example, when the location of a server containing information of global interest is broadcast on national television. The unfortunate result is a sudden overload of the server's network and nearby routers and intolerably long download times. Users often

[1] For clarity of discussion, we assume a constant stream of requests. Clearly, the size of the pool needs to be bigger if the stream of requests is a random process.

exacerbate the problem by reattempting to make connections when connections are not established the first time.

There are many examples of such flash-crowds. In 1994, when the Shoemaker–Levy 9 Jupiter Images were processed in European Southern Observatory (ESO) and posted on the WWW server, the number of accesses per weekday jumped from 2500 to 40 000 and stayed consistently at this value for the entire week during the comet collision. The server's link to the Internet was completely saturated [20]. This is, in fact, a worsening problem as the number of Internet users continues to explode. In 1996, the Netscape homepages were recording more than 80 million hits a day [8]. In June 1997, the sites that were maintaining the Mars Pathfinder pictures saw an aggregate of 160 million hits per day [15]. More recent examples occurred following the release in the Internet of newsworthy reports from the government, applications for free personal computers, and popular movie trailers.

We present a detailed simulation analysis of pseudoserving in dissipating flash-crowds. First, we describe our simulation model and its parameters. Then we discuss the simulations themselves, illustrating the effectiveness and feasibility of pseudoserving under a variety of conditions.

A. Simulation Model

The simulator is a C++ program that takes input describing the conditions of the simulation and produces output logs of important events organized into separate files. The input includes the parameters describing a superserver, parameters describing the superserver's link to the Internet, and parameters describing individual pseudoservers, including the time at which they make their initial requests. The simulator assumes the Internet itself has infinite bandwidth and hence does not model delays caused by congestion at intermediate links.

1) Link Model: We model a link as being composed of two independent portions, an uplink and a downlink. The downlink receives messages from clients and transfers them to the superserver. Similarly, the uplink receives messages from the superserver and transfers them to clients. Each link has a finite bandwidth that affects the rate at which messages are transferred. A link with bandwidth BW bits/s is able to completely transfer a message of M bits in M/BW s. This assumes that the link is not transferring any other messages, and that for the duration of the transfer, no new messages arrive and no pending messages depart.

The link model accounts for the effects of message arrivals and departures by keeping track of message completion times. Using the link bandwidth, the number of messages currently served by the link, and the size of a newly arriving message, two operations are performed when a new message arrives. First, a message completion time is computed for the new message. Second, the message completion times of all pending messages are updated to reflect the additional bandwidth taken by the new message. This is also done whenever a message departs from the link; message completion times are updated to reflect the extra bandwidth released by message departures.

Although this is a simple model, it captures the average load on a link. In a real network, data are transferred as packets from buffer to buffer. Under conditions of heavy traffic, some packets are transferred at the expense of others, in which case, timeouts occur. The simulator, on the other hand, treats all requests with equal priority: a request handed to the link will be put in the link. A nice consequence of this is that the load on a link can be accounted for by simply noting the number of simultaneous messages in the link as a function of time.

TABLE I
FOUR DIFFERENT USER PROFILES FOR THE USERS PARTICIPATING IN THE FLASH-CROWD

Profile	Retry Probability	Retry Delay (seconds)	Request Rate (requests per second)	Duration (seconds)
1	0.995	4	2.0	0, F
2	0.99	3	2.4	0, 0.8348F
3	0.98	4	3.1	0, 0.5555F
4	0.95	5	2.4	0, 0.2593F

2) Network Model: In the simulations, users are connected to the Internet through 28.8-kb/s links and make requests for the same 100-KB file stored on a server, which is connected to the Internet through a 1.544-Mb/s link. Each protocol message, examples of which include request message, reject messages, and referral messages, takes 500 bytes. The concurrency of the server was set to 54, with 53 concurrencies allocated for serving traditional clients, and 1 allocated for pseudoservers. This number was chosen so that when the server is fully utilized, the rate at which data is transferred on any individual connection is 28.8 kb/s.

3) User Model: The parameters of all the simulations were set according to flash-crowd conditions experienced in the telephone network reported in [6]. Although there are clear differences between the telephone network and the Internet, these conditions provide a reasonable starting point for experimentation. Further simulations based on server traces are planned for the future.

The parameters account for user behavior by modeling four types of users with profiles specified by Table I. A few comments should be made regarding these profiles. First, the relative magnitudes of each of the entries in the request rate and duration columns are based on the model used in [6]. The actual values, however, were obtained through experimentation; the values in the request rate column lead to peak rates of requests on par with peak rates of popular WWW sites, on the order of a few hundred requests per second; the values in the duration column lead to plots that clearly show both the transient and steady state behavior of various parameters. F was set to 1800 s in all of the simulations except for the last ones concerned with network clustering, in which F was set to 7200 s. Second, the values of each of the entries in the retry probability and retry delay columns are based on the behavior of telephone users. Although we expect retries to be faster for Web users, we expect them to be significantly less persistent in making the reattempts. The simulation results should be interpreted in light of these comments.

4) Pseudoserving System: The pseudoserving system is the same as the one described in Sections III-A and III-B except for one detail. In the previous description, the superserver keeps track of the IP address of users who agree to contracts.

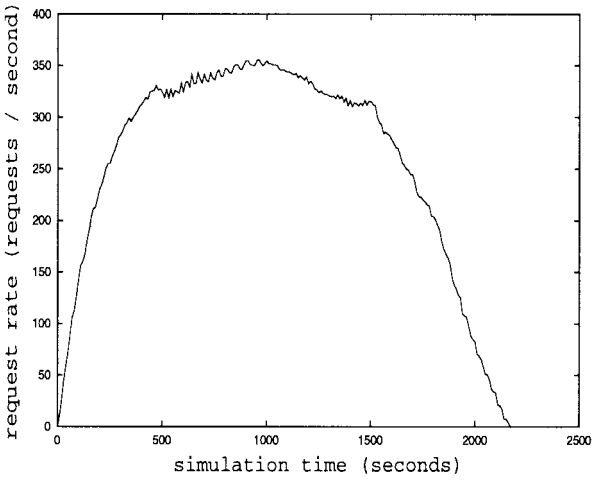

Fig. 2. Request rate under traditional client/server model.

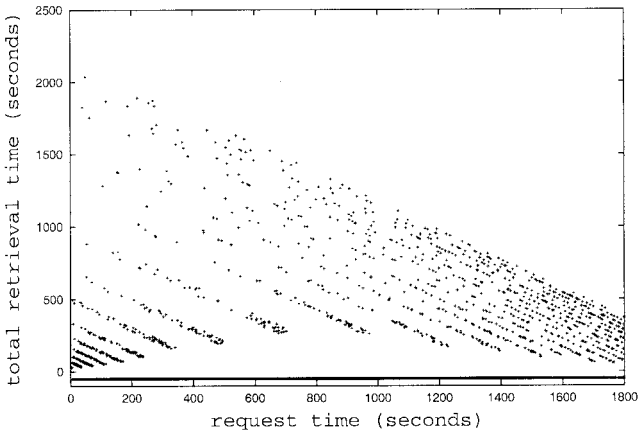

Fig. 3. Total retrieval time under traditional client/server model (give-ups mapped to −50).

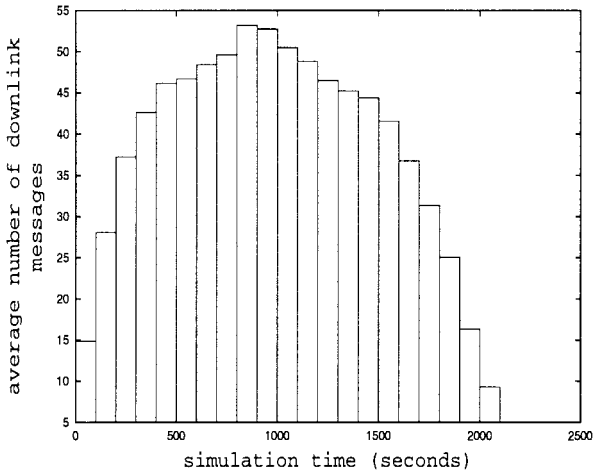

Fig. 4. Number of messages in server's downlink averaged over 100-s intervals.

The superserver in the simulator does not do this; rather, it relies on pseudoservers to report to it when they are ready to serve their files. This modification simplifies the handling of user dishonesty—dishonest users simply do not report back to the superserver.

While the system is essentially the same, the contract policy used in the simulator is quite different. Section III-C describes a policy in which users serve zero, one, or two other users depending on the state of the pseudoserving system. The contract policy used in the simulator, on the other hand, requires that users serve two other users all the time as a condition for obtaining referrals. This more stringent requirement provides resources to buffer a pseudoserving system against the effects of user dishonesty.

B. Results and Discussion

We show the results in four steps. First, we show the effectiveness of pseudoserving in dissipating flash-crowds. Second, we show that pseudoserving is feasible in the sense that the contract can be met by a typical user. Third, we show that pseudoserving is effective even when a large percentage of users are dishonest. Finally, we show the behavior of the pseudoserving system when locality is exploited under realistic patterns of network access.

1) Effectiveness of Pseudoserving: Pseudoserving is effective because it provides the necessary bandwidth to satisfy very high rates of requests. This section compares the gross performance of a pseudoserving system with that of a traditional client/server system.

Fig. 2 shows the rate of request seen by the server of a traditional client/server system. This rate is measured by the server using a window size of 10 s. Notice that while the peak rate of first-attempt requests is 9.9 requests/s (sum of request rates in Table I), the rate seen by the server is actually much larger due to reattempts initiated by users who were rejected. With this request profile, the server is unable to service requests as they arrive after reaching its concurrency limit, which happens on the order of 5 s.

The actual time that it takes for a user to retrieve the file, the *total retrieval time*, depends on two factors. The first factor is competition for access to the server, which depends on the server's concurrency and the number of users competing for access. This fact can be seen in Fig. 3, which shows a plot of the total retrieval time as a function of when a request is made. The retrieval time depends on how fortunate a user is in making a reattempt the moment the server has finished serving a request. But on average, this duration decreases with time as manifested by the decreasing envelope of the plot. Fewer users are competing for access because requests have been satisfied or users have given up. Note that "give-ups" are mapped to −50 in the plot.

The second factor is the bandwidth available for receiving and serving requests. The crucial point to note here is the bandwidth used for sending protocol messages can be significant. While Fig. 4 shows the server's downlink is able to comfortably receive requests at the rate specified in Fig. 2, Fig. 5 shows the uplink is unable to keep pace with the messages it needs to send. In addition to serving 54 concurrent requests, the uplink needs to send reject messages. Unfortunately, the uplink does not have sufficient bandwidth to do so, resulting in an accumulation of messages in the link. It is not until near the end of the simulation that the number of

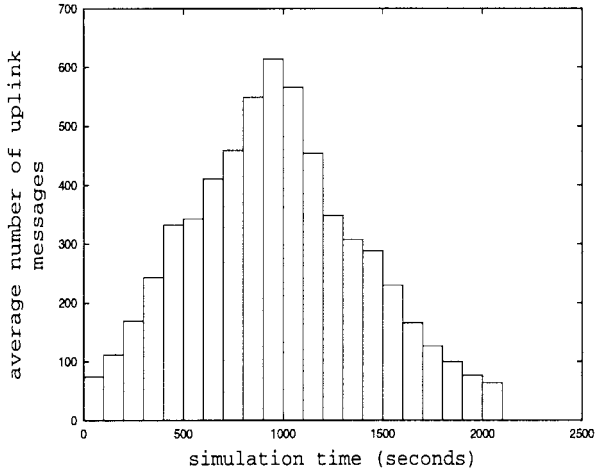

Fig. 5. Number of messages in server's uplink averaged over 100-s intervals.

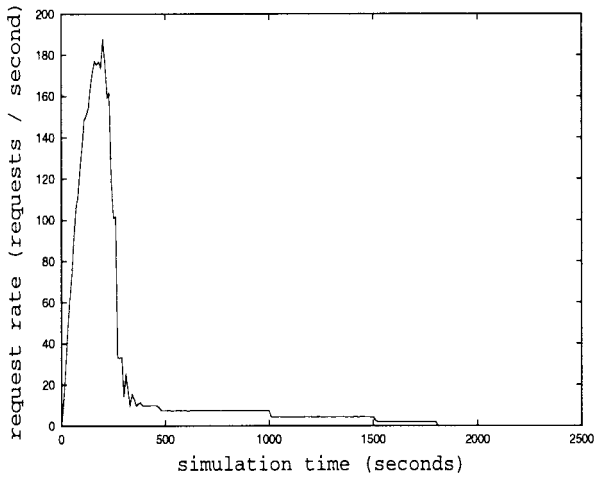

Fig. 6. Request rate under the pseudoserver model.

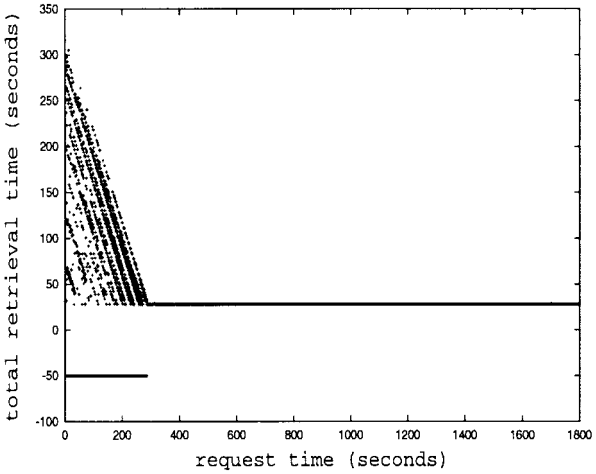

Fig. 7. Total retrieval time under the pseudoserver model (give-ups mapped to -50).

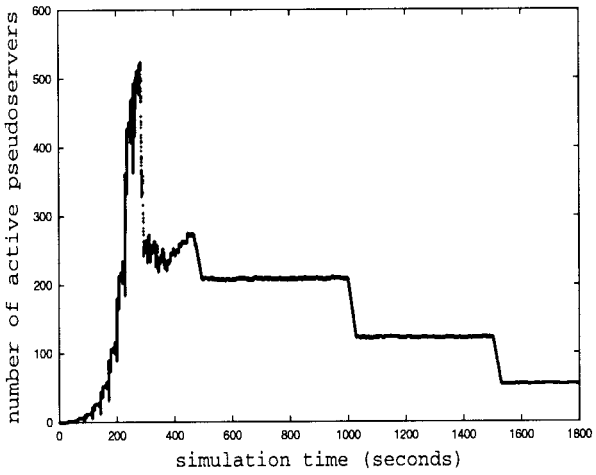

Fig. 8. Number of active pseudoservers.

messages in the link recedes to a number close to the server's concurrency of 54.

Fig. 6 shows the rate of request seen by a superserver. Its shape and scale is clearly far different from the corresponding plot for the traditional client/server system. The peak rate is on the order of 190 requests/s, or roughly half that of the client/server system. Moreover, this peak rate does not dominate nearly as much time as that of the client/server system. In fact, the rate of request for the pseudoserving system plummets to rates on the order of the arrival rate of unique requests by about 470 s into the simulation.

The plot for total retrieval times for the pseudoserving system, shown in Fig. 7, is also far different from the corresponding plot for the client/server system in Fig. 3. The maximum retrieval time is almost a sixth of the maximum retrieval time for the client/server system. Moreover, by about 470 s, the retrieval time reaches the minimum, corresponding to the time it takes to download the file at the rate supported by the modem.

Such dramatic reduction in the total retrieval time is made possible by the extra bandwidth provided by pseudoservers. Fig. 8 shows the number of pseudoservers actively serving requests as a function of time. Note how this number tracks the rate of requests shown in Fig. 6. When the request rate is high, pseudoservers are accumulated to provide extra bandwidth; when the request rate is steady, this number also remains steady; when the request rate drops, this number also drops.

In fact, the number of active pseudoservers is proportional to the request rate. To see this, one can view the pseudoserver pool as an adaptive server with a thruput of R_{out} requests per second. This thruput is equal to the product of the number of pseudoservers actively servicing requests and the rate at which each pseudoserver can service a request. In other words, $R_{\text{out}} = \text{pool size} \times 1/\text{download time}$.

Under steady-state conditions, R_{out} equals the rate at which requests arrive. If we denote this rate by R_{in}, pool size is therefore equal to $R_{\text{in}} \times \text{download time}$. This result is corroborated by the steady-state regions in Fig. 8. Between about 1000–1500 s, for example, the request rate is 4.4 requests/s, as specified by Table I. At 28.8 kb/s, the download time for the 100-KB file is 28 s. Hence, we expect the number of active pseudoservers to be 4.4×28, or 123. Fig. 8 verifies this result. Note that reattempts by users do not skew the request rate in the steady-state because all first-attempt requests are satisfied by the pseudoserving system.

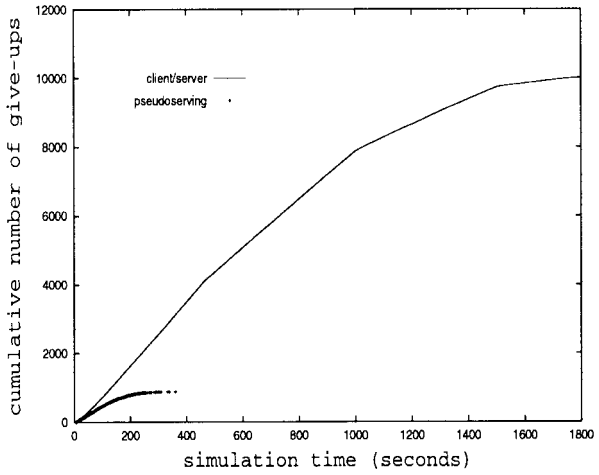

Fig. 9. Comparison of number of give-ups.

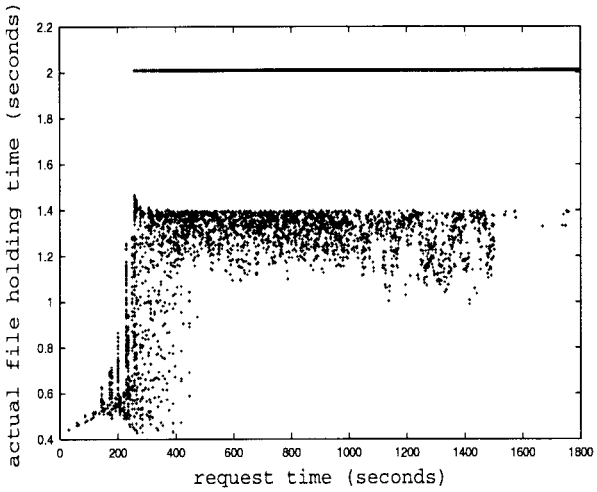

Fig. 10. File-holding time of pseudoservers that served.

An interesting measure of the effectiveness of a scheme in handling flash-crowd situations is the number of users who give up. Give-ups occur as a result of users not being persistent enough in retrying until access is granted. Fig. 9 shows the cumulative number of give-ups derived from Figs. 3 and 7. By the end of the simulation, about ten times as many users give up in the traditional client/server case as there are users who give up in the pseudoserving case. Notice that there are no additional give-ups after 400 s for the pseudoserving case, while give-ups continue to occur for the entire duration of the simulation for the client/server case.

2) Feasibility of Pseudoserving: Recall that a contract entails having the pseudoserver hold the file for a contractual file-holding time, within which it is to serve requests directed to it. If this duration is too large, users may not want to participate in pseudoserving. Hence, it is important to quantify its value. This simulation sheds light on this issue.

In the simulation, the contractual file-holding time was set to 1 s, and all pseudoservers observe a 1-s grace period beyond this agreed-upon file-holding time. Its purpose is to allow a referral to be made on the edge of the contractual file-holding time and still be accepted by the pseudoserver, which might not otherwise have done so due to the nonzero time it takes for the referrals to be sent from the superserver to the pseudoserver. This grace period also conveniently marks those instances when a pseudoserver has reached its contractual file-holding time without having served the maximum number of requests.

Fig. 10 shows the actual file-holding time of each pseudoserver in the simulation. The horizontal coordinate of each dot corresponds to the time when the pseudoserver has gotten its requested file and is ready to serve. The length of time transpired between this point and when it receives its second request to serve is represented by the dot's vertical coordinate.

From this plot, one can see there is no file-holding time greater than 2 s, corresponding to the 1-s contractual file-holding time plus another second for the grace period. Most of the file-holding time reside between 1–1.4 s. Hence, for this simulation in which the size of protocol messages is set to 500 bytes, a nonzero grace period is important. A significant amount of time is spent transferring referral messages to clients.

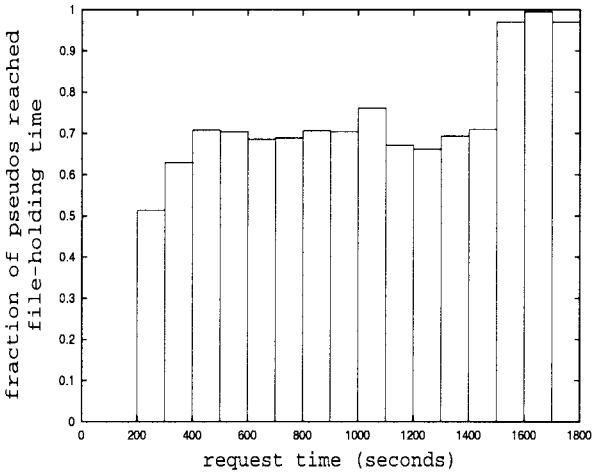

Fig. 11. Fraction of pseudoservers that reached contractual file-holding time over 100-s intervals.

Fig. 11 shows the fraction of pseudoservers that held onto their files for the maximum file-holding time plus the grace period. This fraction is zero near the beginning of the simulation because pseudoservers fulfill their contract by serving two users long before they hold their files for the contractual file-holding time. As the number of pseudoservers grow beyond what is necessary to satisfy the stream of requests, more pseudoservers hold their files for the contractual file-holding time. The figure in fact shows that more than 50% of the pseudoservers created belong to this category after user-initiated retransmissions stop near 300 s into the simulation. This suggests that a smart superserver that dynamically changes the file-holding time according to request patterns can significantly reduce the average length of time pseudoservers need to hold their files, thus making it easier for hosts to participate as pseudoservers.

3) Robustness of Pseudoserving Under Realistic User Behavior: Because pseudoserving depends on the promise of users to satisfy contracts, and not all users are honest, it is

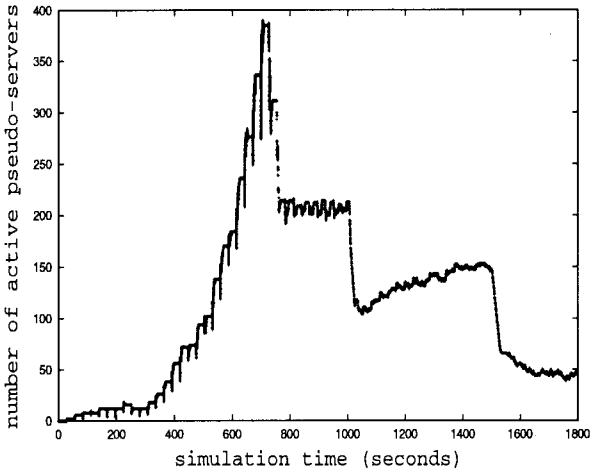

Fig. 12. Number of active pseudoservers.

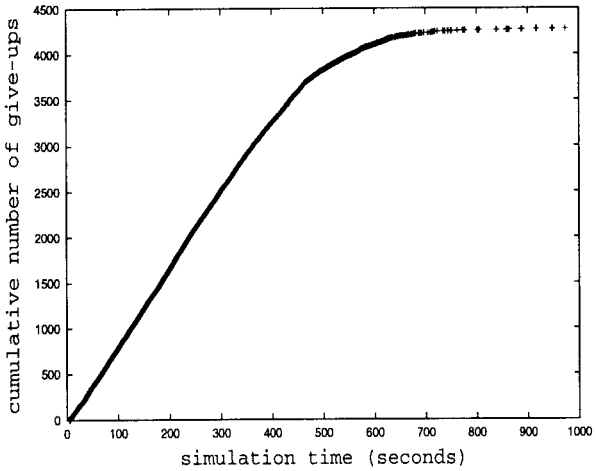

Fig. 13. Cumulative number of give-ups for 40% dishonesty.

important to address the issue of how sensitive pseudoserving is to breaches of contracts. Under the simulator's particular implementation of pseudoserving, in which all pseudoservers are required to serve two other users within a file-holding time, the answer is simple: on average, there will always be growth in the number of pseudoservers if more than half of the users satisfy their contracts. To see this, suppose there are N pseudoservers, and $N/2$ of them breach their contracts by not serving any other user. In the next iteration, the $N/2$ honest pseudoservers will generate $2 * (N/2)$, or N, new pseudoservers, so that in next iteration after that, there will still be N pseudoservers. The $N/2$ honest users are in a sense "subsidizing" the $N/2$ dishonest users. The expected number of pseudoservers will always grow if user dishonesty is less than 50%.

These points are illustrated by Figs. 12 and 13, which plot the number of active pseudoservers and the number of give-ups as a function of time for the case of 40% user dishonesty. Even with user dishonesty of this magnitude, pseudoserving is still quite effective, as can be seen in Fig. 13. Note that beyond about 700 s into the simulation, which coincides with the time where the peak number of active pseudoservers occurs, there are very few give-ups. This is an important point. It means under flash-crowd conditions, the detrimental effects of user dishonesty of up to 50% is limited to only the initial growth phase of the pseudoserver pool. Beyond that, user dishonesty does not affect the retrieval time for new users because the honest users and the superserver "subsidize" the dishonest users by serving requests not served by them.

4) Feasibility of Pseudoserving Under Realistic Network Access Patterns: So far, referrals were made without regard to the relative locations of the requester and the pseudoserver serving the request. Clearly, it makes sense to ensure referrals are made to the pseudoserver closest to the requester. Everything else being equal, files transfers that cross fewer links are faster and generate less network traffic. They are therefore beneficial both from the standpoint of the individual user, who wishes to reduce latency associated with file transfers, and from the standpoint of the user community, whose members all benefit when fewer links are crossed for each transfer.

One way to ensure that a requester is served by a pseudoserver close to it is to make referrals only to the same network as the requester. The superserver can do this by storing each member of the pseudoserver pool in a data structure indexed by the member's network address. Transfers based on referrals are then ensured to be within the same network by making referrals only when a pseudoserver exists in the same network as the requester. However, there is a cost in implementing this policy; the contract needs to be set so that the file-holding time is sufficiently long. In particular, it must be long enough so that a request arrives from a network before the file-holding time expires for all the pseudoservers in that network.

But ensuring referrals are made to the same network may be too inflexible; requests for the same file on a superserver from users on the same network may come too few and far in between. As a result, the file-holding time may not be easily satisfied by a typical user who is connected to the Internet only temporarily. Rather than ensuring network locality, it is more feasible from the standpoint of setting file-holding times reasonably satisfiable by pseudoservers to ensure transfers are made within groups of nearby networks, or *network clusters.*

We address the issue of how referrals can be made to network clusters only briefly here. In principle, the superserver can form such network clusters based on *a priori* information regarding the location of each network and the number and types of links connecting them (we do not consider mobility here). A host's network ID can then be used to determine the network cluster to which the host belongs. Another possibility would be to have the client send its "network cluster address" along with its request. By encoding topological information, such an address would obviate the need for the server to keep a vast table of network addresses and their relative topological locations.

We used statistics reported in [2] as the basis for setting parameters in the simulation. In particular, a typical server is accessed by thousands of domains, 10% of which account for 75% of the requests. We assumed a server is accessed by 5000 domains, 500 of which account for 75% of the requests *(hot*

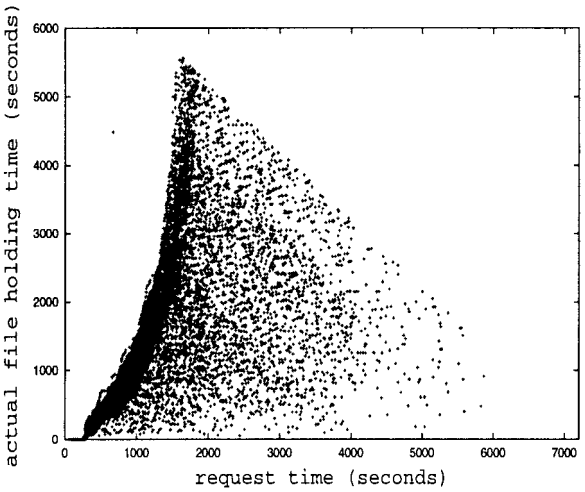

Fig. 14. File-holding time of pseudoservers in the hot domain.

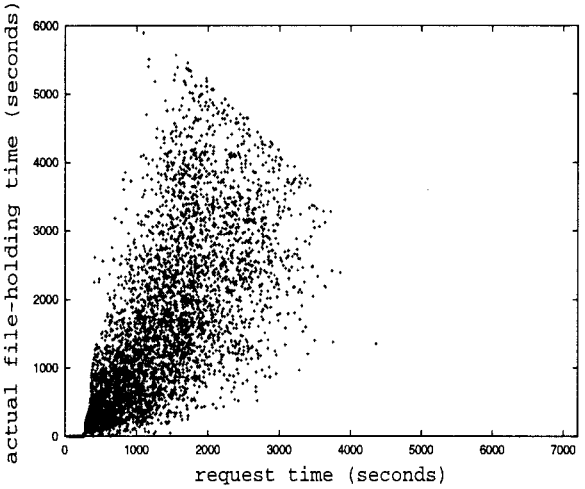

Fig. 15. File-holding time of pseudoservers in the cold domain.

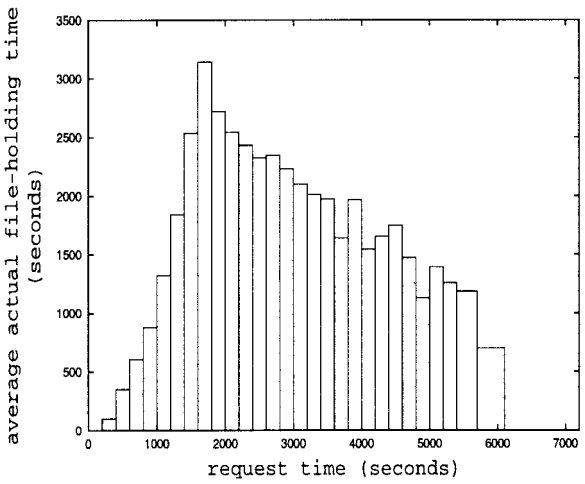

Fig. 16. File-holding time of pseudoservers in the hot domain averaged over 200-s intervals.

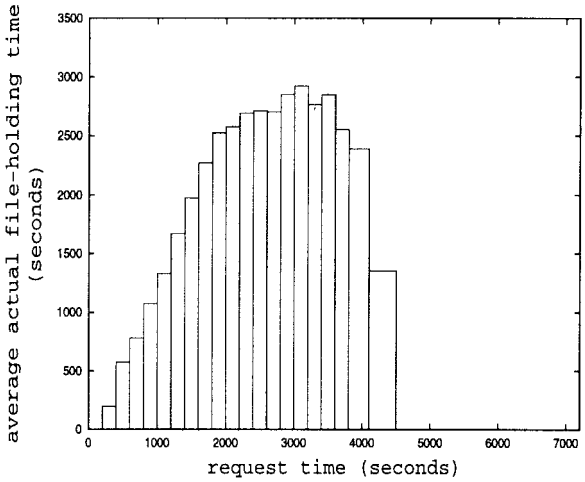

Fig. 17. File-holding time of pseudoservers in the cold domain averaged over 200-s intervals.

domains) and 4500 of which account for the remaining 25% of the requests *(cold domains).*

To test whether pseudoserving on a per cluster basis is feasible, we set the contractual file-holding time to a very large number (100 000 s) and observed the actual file-holding time for the various cases of hot and cold domains. The network cluster size was set to ten, so that each of the clients from the 5000 networks that accessed the superserver came from one of 500 network clusters.

Figs. 14–17 show plots of the actual file-holding time as a function of when requests are made. Before we begin the discussion, it is important to note that these plots show only the cases in which a pseudoserver has served *two* referrals. Pseudoservers that served less than two times had to hold onto their files for the entire duration of the contractual file-holding time. These cases were filtered out of the plots for the purpose of clarifying the discussion to follow.

First, the figures show that even for the simple contract policy in which all pseudoservers are required to serve two other users within a specified file-holding time, the longest time that a pseudoserver actually needs to hold is on the order of 5600 s, or about an hour and a half, for the hot domain case. The peak average time using a window size of 100 s is about half as long, on the order of 50 min.

As expected, the file-holding time is very small at the beginning of the simulation. There are far more requests than there are pseudoservers to handle them. But within about 250 s, more than enough pseudoservers are generated to handle the requests. As a result, pseudoservers have to hold on to their files for a longer period of time before a referral is directed to them. This is evident in Fig. 14 in which the average file-holding time grows with the time of request, up to about 1700 s.

Somewhat surprising is that for much of the early part of the simulation, the file-holding time of hot domains is actually *larger* than that of the cold domains. The reason for this relates to the exponential nature by which pseudoservers are created. In the case of the hot domains, many more excess pseudoservers are created than are created in the cold domains. As a result, there is more competition for new requests in the hot domains than in the cold domains, and the average file-holding time is thus larger for the hot domains.

These simulations show that there is clearly much to be gained by setting contracts more intelligently. In particular, when the rate of request is low on a per-domain basis, contracts should be set so that pseudoservers do not grow in number; doing so creates unnecessary pseudoservers and causes the file-holding time to increase beyond what is necessary.

V. Experiments

To further investigate the effectiveness of pseudoserving, we tested a prototype system on a LAN of 113 workstations. One of the machines was used to run the superserver process, while a subset of the others were used to run pseudoserver processes. We conducted a series of experiments in which each pseudoserver requested the same file from the superserver over a period of several minutes. This section discusses the details of these experiments and their results.

This preliminary work serves two important purposes. First, it brings pseudoserving from the realm of ideas and simulations onto firmer ground; factors not accounted for in previous work on pseudoserving are accounted for in the experiments. These include limited disk to memory bandwidth, nonzero processing times, overhead of lower level protocols, limited backbone bandwidth, and network topology. The latter two are especially important and are considered later in this section.

The prototype system is also the first step toward the integration of pseudoserving into the WWW. It is important that pseudoserving works in conjunction with current Web technologies, in particular, caching mechanisms. Toward this end, we would like to implement the pseudoserver as a module in an existing cache framework, and the code developed in the prototype system serves as a starting point for this purpose.

A. Overview

The experiments were run on workstations within the Department of Computer Science during the summer, outside the regular academic year. These machines are a collection of 64 Dec 5000's running Ultrix 4.4, 25 HP 712's running HP-UX 10.1, and 24 SGI Indy's running IRIX 5.3 connected to each other through a standard shared-medium (nonswitched) 10-Mb/s Ethernet LAN. Casual surveys show that only about 10% of the machines are used in a typical summer day.

The scope of the experiments was constrained by the state of these machines at various times of the day and at various days of the week. Many of the machines were down for maintenance reasons as well as day to day breakdowns due to imperfect software running in a heterogeneous environment. A further constraint was imposed by the number of processes that could be run and the free space that was available on the local disk of each machine.

A `process spawner` was written to exploit the resources available. It takes as input user-specified parameters concerning the number of processes to spawn and the machines on which to spawn them. It then probes the specified machines individually to determine the number of processes each can run, mindful of the file size and the disk space available on the machine. Next, it spawns the pseudoserver processes and synchronizes their local clocks. Finally, it instructs each process to retrieve a specified file from the superserver at a certain time and to report important statistics regarding retrieval times to a `statistics collector`. The `statistics collector` itself as well as the `superserver` were run manually.

The manner in which pseudoservers were spawned has important implications on locality. Pseudoservers are spawned on each machine in a round-robin fashion. For experiments in which the number of pseudoservers desired exceeds the number of machines, at least one machine runs more than one process. Typically, on the order of ten processes ran on each of about a hundred machines so that about one percent of the requests were satisfied from the same host when pseudoserving is active. As will be discussed later, a parameter can be set to ensure that requests are satisfied either by the superserver or by a pseudoserver residing on the same machine as the requester. When this parameter, which specifies locality, is set, the percentage of requests satisfied from the same machine rises to the order of 90%.

The `superserver` is an implementation of the superserver described in Section IV-A-4 with some important differences noted below.

- Instead of responding with either a referral or a refusal, the superserver responds with either a referral or a notice to put the requester in *waiting mode.* When the latter happens, the superserver stores the requester's IP address and the requester, in turn, waits to be served by a pseudoserver. This modification has two effects. It has the negative effect of increasing the storage requirements of the superserver. Each request requires extra storage space in the superserver's main memory. It has the positive effect of discouraging reattempts initiated by the user because reattempts do not hasten the establishment of connections; reattempts simply put users further behind in a waiting queue.[2]

  Although this queuing mechanism could be implemented on traditional servers, it is much more appropriate to do so on superservers. Under pseudoserving, requests on the waiting queue are satisfied quickly because resources are created exponentially to fulfill them. Were such a mechanism implemented on traditional servers, the server could run out of memory very quickly because the rate at which requests arrive is much higher than the rate at which they can be served.
- To simplify the design of the superserver, the superserver does not keep track of the period of time that each pseudoserver has kept its file. Instead, the superserver relies on the pseudoserver to report when it has finished retrieving the file and is ready to serve. This adds an extra connection to the superserver for each request.
- The superserver responds to a request to service by telling the pseudoserver that it need not serve, it needs to serve a client in waiting mode, or it needs to be put in *standby*

[2]This assumes the superserver serves waiting queues on a first-come-first-serve basis. With locality turned off, that is precisely what happens. With locality turned on, requests are satisfied on a first come first serve basis within the same network cluster.

mode. In the last case, the pseudoserver becomes part of the pool of pseudoservers waiting to service referrals.

The `pseudoserver` is the same as the one described in Section III-A. There are three important points to note. First, in all the experiments in which the pseudoserver operated in client/server mode, the pseudoservers were set so that they continue to make requests until the file is retrieved. Second, user dishonesty is not accounted for, i.e., the pseudoservers are always cooperative. Finally, pseudoservers send important statistics concerning their operation to the `statistics collector`, which keeps track of important events reported by each pseudoserver as it is run in the experiments. These statistics are:

Connection Time: Amount of time that transpired since the first request is made until the pseudoserver has either been told to retrieve the file directly from the superserver, given a referral to where the file can be retrieved, or served by a host requesting to serve a pseudoserver.

Average Waiting Time: (AWT) is the average of the total waiting time which includes the connection time and the time it takes to download the requested file.

Source IP: The host address from which a client retrieves its file. The percentage of requests from the superserver, from the same host, and from other hosts are designated as SS%, SH%, and OH%, respectively.

Standby Time: Amount of time transpired since the pseudoserver has completed its download until it has either fulfilled its contract by serving two clients or holding onto the retrieved file for the contractual file holding time.

B. Experimental Parameters

The parameters of the process spawner define the experiments. Some of these parameters were common to all of the experiments while others were different. The common ones include:

Interarrival Time: Once the `process spawner` has spawned the pseudoserver processes, it notifies each of them when to make its first request. This parameter determines the time between consecutive first requests and is set to 1 s.

Concurrency: This is the superserver's concurrency. It is set to ten.

Size of Pseudoserver Pool: Determines the superserver's response to requests to serve sent by the pseudoserver. If the size of the current pool of pseudoservers is smaller than this parameter, the pseudoserver is added to the pool. Otherwise, the pseudoserver is told that it need not serve. This parameter is set to 100.

The experiments were differentiated by setting the following parameters:

Processes Wanted: Number of pseudoserver processes that is requested for a particular experiment. The actual number of processes spawned, designated PROCS, depends on the state of the machines.

File Size: Designated FS, is the size of the file to be pseudoserved.

Pseudoserving: Designated PS?, is on if pseudoserving is turned on. Otherwise, the superserver operates as a traditional server.

TABLE II
EXPERIMENTAL RESULTS

Exp #	FS (bytes)	PROCS	PS?	FHT* (s)	LCL?*	SS%*	SH%*	OH%*	AWT (s)
1	1,000	1780	off	N/A	N/A	N/A	N/A	N/A	0.85
2	1,000	1656	on	300	off	0	2	98	1.1
3	1,000	1714	on	300	on	5	95	0	0.93
4	10,000	1801	off	N/A	N/A	N/A	N/A	N/A	0.99
5	10,000	1800	on	300	off	0	1	99	1.1
6	10,000	1721	on	300	on	6	94	0	1.1
7	100,000	1800	off	N/A	N/A	N/A	N/A	N/A	1.3
8	100,000	1801	on	300	off	0	1	99	1.3
9	100,000	1705	on	300	on	6	94	0	1.2
10	1,000,000	1511	off	N/A	N/A	N/A	N/A	N/A	391
11	1,000,000	1513	on	300	off	0	2	98	4.8
12	1,000,000	1448	on	300	on	10	90	0	2.4
13	3,000,000	966	off	N/A	N/A	N/A	N/A	N/A	1628
14	3,000,000	909	on	300	off	17	2	82	346
15	3,000,000	830	on	300	on	13	87	0	8.5
16	10,000,000	608	off	N/A	N/A	N/A	N/A	N/A	3855
17	10,000,000	952	on	300	off	7	3	91	1399
18	10,000,000	365	on	300	on	11	89	0	149

File Holding Time: Designated FHT, is the contractual file holding time.

Locality: Designated LCL?, is on if requests are satisfied only either directly by the superserver or by a pseudoserver located in the same host as the requester. Otherwise, requests are satisfied regardless of the relative location of the requester and the requestee.

C. Experimental Results

The experiments are described in Table II. Each row represents an experiment. The first five columns show the parameters of the experiment, and the others show the results of running it.

The experiments can be understood by first noting that they are grouped according to file sizes of 1 KB, 10 KB, 100 KB, 1 MB, 3 MB, and 10 MB, and that the average waiting time increases accordingly. Also note that each of these groups contains three experiments. The first corresponds to the case where pseudoserving is turned off; the second corresponds to the case where pseudoserving is turned on but locality is turned off; and the third corresponds to the case where both pseudoserving and locality are turned on.

The average waiting times are what we expect from pseudoserving operating under the different regimes. When the file is small, server concurrency and bandwidth is not an issue and the user sees little difference in retrieving files from a traditional server or from a superserver. In fact, retrieving from a superserver takes slightly longer due to the overhead associated with implementing the pseudoserver protocol. When the file is large, both server concurrency and bandwidth become bottlenecks and we see pseudoserving working effectively by reducing the total retrieval time several-fold. When transferring a 3-MB file, for example, the retrieval time is reduced from about half an hour to about 5 min. When locality is turned on, the transfer time is reduced to less than 10 s.

While the LAN within which the experiments were run is not the Internet, the experiments demonstrate first order effects that applies to arbitrary networks including the Internet. The Ethernet connecting many hosts can be likened to an Internet backbone connecting many network gateways. While it is clear

that pseudoserving can be used to bypass server concurrency, the experiments show that significant gains can be had by bypassing the backbone as well. Experiments 14 and 15, for example, show a reduction in the retrieval time by a factor of 40 when the "Ethernet backbone" was bypassed. Clearly, we do not expect transfers to take place from within the same host as it is done in the experiments. At the same time, we do expect significant improvements in the retrieval time when referrals are made to the same network cluster as the requester.

An earlier version of the system was actually run with the superserver behind a 28.8-kb/s modem. The pseudoservers themselves were connected to each other through a 10-Mb/s LAN. Not surprisingly, the time it took to retrieve a 100-KB file was reduced from several minutes to essentially zero once one of the pseudoservers retrieved the requested file from the superserver. Nevertheless, this simple experiment is a very real demonstration of the power of pseudoserving: by shifting the burden of distributing data to the clients, pseudoserving enables content providers to distribute their content very cheaply.

VI. Limitations

We discuss the boundaries of pseudoserving in this section. These include the conditions under which a pseudoserving system functions effectively and the new bottleneck that arises when pseudoserving is implemented.

A. Environments Suitable for Pseudoserving

Pseudoserving functions most favorably under two conditions: 1) when everyone wishes to retrieve the same file and 2) when the file size is large. If users retrieve many different files from a server, sharing files becomes more difficult. If the file size is small, the overhead of the pseudoserving protocol may outweigh the benefits of pseudoserving. We discuss these considerations in the following subsections.

1) Pseudoserving Multiple Files: Although pseudoserving works well when the file requested is the same for everyone, this is not a necessary condition. Pseudoserving can be applied on a per-file basis in which referrals are made only to pseudoservers that contain the requested file. If the rate of request for each file is sufficiently high, pseudoserving on a per-file basis works equally well.

A problem arises if the rate of request is high, but the rate of request for individual files is low. Recall that when the number of pseudoservers equals `request rate` $\times$ `download time`, a pseudoserver becomes available just as a new request arrives.

Consider the situation in which `request rate` $\times$ `download time` < 1. This can happen, for example, when a superserver contains many files but requests for any individual file arrive few and far in between. Under such circumstances, the contractual file holding time must be set sufficiently long so that pseudoservers are available when referrals are directed to them.

The problem, then, is as follows. Applying pseudoserving on a per-file basis effectively reduces the rate at which requests are made. For sufficiently large reductions, pseudoservers must make their files available to requesters for a longer period of time. This may not be acceptable for users who are, for example, connected to the Internet only temporarily. It should be noted that this is the same problem described in Section IV-B-3 in which pseudoserving is applied in a per-network-cluster basis.

TABLE III
Bandwidth Saving With Pseudoserving for Different File Sizes

Document Type	Typical Document Size in Bytes	Bandwidth Savings
text-based web page	2,000	4×
image	10,000	20×
mpeg video	2,000,000	4,000×
shareware	5,000,000	10,000×
linux distribution	100,000,000	200,000×

2) Effectiveness of Pseudoserving According to File Size: Pseudoserving shifts the role of the server from serving files to one of serving pointers to where files can be retrieved. Hence, bandwidth usage on the server's link to the Internet is reduced by a factor of `filesize/pointersize`.

Most of the bytes of a pointer are taken by the overhead associated with protocols at the transport and lower layers. Assuming the TCP/IP protocol, for example, it takes seven packets to set up and terminate a connection. Each of these packets takes 40 bytes, for a total of 280 bytes. Using the implementation of the superserver described in Section III-B, a referral is simply a 4-byte IP address plus a checksum for the file.

We assume conservatively each pointer uses 500 bytes. With this pointer size, the savings in server bandwidth under pseudoserving are shown in Table III.

Even for text-based web pages, there is roughly a four-times reduction in the usage of valuable server bandwidth. Pseudoserving therefore significantly reduces the number of bytes the server transfers even when the file is small. Clearly, when the files are large, the savings become tremendous.

Note that the reduction in the number of bytes that the server transfers does not necessarily translate to a corresponding reduction in the total retrieval time. Pseudoserving requires that a client establish a second connection with a pseudoserver after having established a first one with the superserver. This overhead often becomes more visible to the user when the file to be transferred is small, as experimental results show in Section V. But when the network is heavily congested, when data dribble to users at a rate of tens of bytes per second, bandwidth again becomes an issue even for the transfer of small files.

B. Superserver Bottleneck

As Section II stated, once the bottleneck due to server concurrency is removed, the server's link to the Internet becomes the new bottleneck; data can be transferred to clients only as fast as the link is able to move them. Similarly, a superserver can only process messages as fast as they arrive from the clients through the link. To a first order, pseudoserving reaches its limit for effectively handling arbitrary rates of requests when the rate at which requests are made exceeds the rate at which they can be processed by the server's link to the Internet.

TABLE IV
LIMIT OF PSEUDOSERVING FOR DIFFERENT TYPES OF LINKS

Link Type	Link Bandwidth (Kbps)	Pseudoserving Limit (requests per sec.)
14.4 Kbps modem	14.4	3.6
28.8 Kbps modem	28.8	7.2
56.6 Kbps modem	56.6	14
ISDN	128	32
T1	1544	386
T3	45000	11250

We can characterize this new limit by assuming the size of requests to be 500 bytes each, accounting for the overhead involved in setting up a connection as we did earlier for the pointer size. The limit for pseudoserving is therefore `bandwidth of link/500 bytes`. Table IV shows this limit as applied to typical server links.

VII. CONCLUSION

Pseudoserving is a new paradigm for accessing information in the Internet that allows its participants to bypass congestion at the server by combining elements of caching and economics. A referral may be obtained to where the requested file may be found if the requester agrees to serve the retrieved file to a small number of users within a short period of time.

Simulations based on the behavior of users under flash-crowd conditions demonstrate the effectiveness and feasibility of pseudoserving in dissipating flash-crowds under a variety of conditions. In particular, they show the number of pseudoservers grows to meet even very high demand. Within about 5 min, for example, enough pseudoservers are generated to reduce the total retrieval time to its minimum. As a result, not only are new requests serviced over an order of magnitude faster, but the number of users who give up because they were unable to obtain a connection to the server after many reattempts is reduced by about ten times. Pseudoserving is also robust against user dishonesty. In particular, even with 40% of the users breaching their contracts, the number of pseudoservers still grows. As a result, the total retrieval time can still be driven to its minimum, albeit at a slower rate. The number of give-ups even with 40% user dishonesty is less than half that associated with a client/server system. Finally, simulations show that pseudoserving is feasible, even under fairly conservative conditions. In particular, when referrals are made only to pseudoservers that are in the same network cluster as the requester, the peak actual file-holding time is on the order of two hours and the average file-holding time is about 40 min. It is important to note that these durations were derived using a simple-minded contract policy in which all users are required to serve two other users, regardless of the rate of request. Policies that account for the rate of request are expected to perform much better.

An implementation of a pseudoserving system has also been developed and tested on a LAN of over a hundred workstations. Experimental results show that pseudoserving reduces the total retrieval time of large files by over an order of magnitude by allowing clients to bypass limits placed by the server's concurrency. They also show that further, significant reductions are possible by exploiting network locality.

Future work revolves around experiments we plan to perform on another implementation of the superserver. We are in the process of enhancing an Apache WWW server with pseudoserving capabilities. After this work is completed, we wish to explore two areas. The first is the performance of the superserver under extreme loads. We are especially interested in the overhead of the pseudoserving protocol and how it impacts the performance of the superserver under different loading conditions. We are also interested in wide-area experiments that shed light on the effectiveness of pseudoserving in exploiting network locality. The results from this work promise both to be interesting and to have practical implications for pseudoserving.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers and the editor, Dr. S. Pink, for their reviews. The reviews were helpful in preparing the final version of the paper.

REFERENCES

[1] M. Abrams, C. Standridge, G. Abdulla, S. Williams, and E. A. Fox, "Caching proxies: Limitations and potentials," in *Proc. 4th Int. World Wide Web Conf.*, Boston, MA, Dec. 1995.

[2] M. F. Arlitt and C. L. Williamson, "Web server workload characterization: The search for invariants," in *Proc. 1996 ACM SIGMETRICS Int. Conf. Measurement and Modeling of Computer Systems*, Philadelphia, PA, June 1996, vol. 24, no. 1, pp. 126–137.

[3] M. Baentsch, G. Molter, and P. Sturm, "Introducing application-level replication and naming into today's Web," *Comput. Networks ISDN Syst.*, vol. 28, pp. 921–930, 1996.

[4] A. Bestavros, R. L. Carter, and M. E. Crovella, "Application-level document caching in the Internet," Comput. Sci. Dept., Boston Univ., Boston, MA, Tech. Rep. BU-CS-95-002, Mar. 1995.

[5] J. Bolot and P. Hoschka "Performance engineering of the World Wide Web: Application to dimensioning and cache design," in *Proc. 5th Int. World Wide Web Conf.*, Paris, France, May 1996, pp. 1397–1405.

[6] J. E. Burns and D. Ghosal, "Automatic detection and control of media stimulated focussed overloads," in *Proc. Int. Teletraffic Congress*, Washington, DC, June 1997, pp. 889–900.

[7] M. D. Dahlin, R. Y. Wang, T. E. Anderson, and D. A. Patterson, "Cooperative caching: Using remote client memory to improve file system performance," in *Proc. 1st Symp. Operating System Desing and Implementation (OSDI)*, 1994, pp. 267–280.

[8] S. L. Garfinkel, "The wizard of netscape," *WebServer Mag.*, pp. 58–64, July/Aug. 1996.

[9] GVU's 4th, 5th, 6th, 7th, and 9th WWW User Survey, Oct. 1995, Apr. 1996, Oct. 1996, Apr. 1997, Apr. 1998. [Online]. Available HTTP: http://www.cc.gatech.edu/gvu/user_surveys

[10] J. Gwertzman and M. Seltzer, "The case for geographical push-caching," in *Proc. of the 1995 Workshop on Hot Operating Systems*, 1995, pp. 51–555.

[11] HTTP 1.1 Specification. [Online]. Available HTTP: http://www.cis.ohio-state.edu/htbin/rfc/rfc2068.html

[12] K. Kong and D. Ghosal, "Pseudo-serving: A user-responsible paradigm for Internet access," in *Proc.6th Int. WWW Conf.*, Santa Clara, CA, Apr. 1997, pp. 546–557.

[13] Low-bandwidth X. [Online]. Available HTTP: http://www.thphy.uni-duesseldorf.de/kielhorn/xfaq/X-FAQ_toc.html

[14] R. Malpani, J. Lorch, and D. Berger, "Making world wide web caching servers cooperate," in *Proc. 4th Int. World Wide Web Conf.*, Boston, MA, Dec. 1995.

[15] Mars Path Finder. [Online]. Available HTTP: http://mpfwww.jpl.nasa.com/index.html

[16] J. K. Mackie-Mason and H. R. Varian, "Some FAQS about usage-based pricing," in *Internet Economics*, L. W. McKnight and J. P. Bailey, Eds. Cambridge, MA: MIT Press, 1997.

[17] L. W. McKnight and J. P. Bailey, "An introduction to Internet economics," in *Internet Economics*, L. W. McKnight and J. P. Bailey, Eds. Cambridge, MA: MIT Press, 1997.

[18] J. K. Mackie-Mason and H. R. Varian, "Pricing congestible network resources," Univ. of Michigan, Nov. 1994.
[19] ——, "Pricing the Internet," in *Public Access to the Internet*, B. Kahin and J. Keller, Eds. Englewood Cliffs, NJ: Prentice-Hall, 1995.
[20] F. Murtagh, "Data on the number of access to the WWW Server in ESO (European Southern Observatory)," private communication, May 1994.
[21] D. Neal, "The harvest object cache in New Zealand," *Comput. Networks ISDN Syst.*, vol. 28, pp. 1415–1430, 1996.
[22] M. Parsa and J. J. Garcia-Luna-Aceves, "Scalable Internet multicast routing," *IEEE J. Select. Areas Commun.*, vol. 15, pp. 316–331, Apr. 1997.
[23] S. Paul, K. Sabnani, J. Lin, and S. Bhattacharyya, "Reliable multicast transport protocol (RMTP)," *IEEE J. Select. Areas Commun., Special Issue on Network Support for Multipoint Communication*, vol. 15, pp. 407–421, Apr. 1997.
[24] S. Shenker, D. Clark, D. Estrin, and S. Herzog, "Pricing in computer networks: Reshaping the research agenda," *Telecommunications Policy*, vol. 20, no. 3, pp. 183–201, 1996.
[25] N. G. Smith, "The UK national web cache—The state of the art," *Comput. Networks ISDN Syst.*, vol. 28, pp. 1407–1414, 1996.
[26] W. R. Stevens, *TCP/IP Illustrated Volume 1: Protocols.* Addison-Wesley, 1994.
[27] ——, *TCP/IP Illustrated Volume 3: TCP for Transactions, HTTP, NNTP, and UNIX Domain Protocols.* Reading, MA: Addison-Wesley, Professional Computing Series, 1996.
[28] B. Laurie and P. Laurie, *Apache: The Definitive Guide*. Sebastopol, CA: O'Reilly, Inc., Mar. 1997.

Keith Kong received the B.S. degree in electrical engineering from the University of California at Berkeley in 1993. He is currently working toward the Ph.D. degree at the University of California at Davis.

His current research interests are in the area of caching and high-level network protocols and their application in mitigating congestion in wide area networks.

Dipak Ghosal received the B.Tech degree in electrical engineering from the Indian Institute of Technology, Kanpur, India, in 1983, the M.S. degree in computer science from the Indian Institute of Science, Bangalore, India, in 1985, and the Ph.D. degree in computer science from the University of Southwestern Louisiana, Lafayette, in 1988.

From 1988 to 1990, he was a Research Associate at the Institute for Advanced Computer Studies, University of Maryland at College Park. From 1990 to 1996, he was a Member of Technical Staff at Bell Communications Research/Bellcore, (presently Telecordia Technologies), Red Bank, NJ. Currently, he is an Associate Professor in the Department of Computer Science, University of California at Davis, His research interests include control and management of high-speed networks, personal communication services, and performance evaluation of computer and communication systems.