DIMACS Technical Report 2000-47
February 2001

An Integrated Architecture for The Scalable Delivery of Semi-Dynamic Web Content

by

Danny Dolev
Institute of Computer Science
Hebrew University
Jerusalem, Israel
dolev@cs.huji.ac.il

Osnat Mokryn[1]
Institute of Computer Science
Hebrew University
Jerusalem, Israel
osnaty@cs.huji.ac.il

Yuval Shavitt[2,3]
Bell Labs
101 Crawfords Corner Rd.,
Holmdel, NJ 07733, USA
shavitt@ieee.org

Innocenty Sukhov
Institute of Computer Science
Hebrew University
Jerusalem, Israel
kesha@cs.huji.ac.il

[1]Corresponding author.
[2]Permanent DIMACS Member
[3]The author is currently also with Dept. of Elecrtical Engineering – Systems, Tel Aviv University, Ramat Aviv 69978, Israel.

DIMACS is a partnership of Rutgers University, Princeton University, AT&T Labs-Research, Bell Labs, NEC Research Institute and Telcordia Technologies (formerly Bellcore).

DIMACS was founded as an NSF Science and Technology Center, and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

The competition on clients attention requires sites to update their content frequently. As a result, a large percentage of web pages are semi-dynamic, i.e., change quite often and stay static between changes. The cost of maintaining consistency for such pages discourages caching solutions. We suggest here an integrated architecture for the scalable delivery of frequently changing hot pages. Our scheme enables sites to dynamically select whether to cyclically multicast a hot page or to unicast it, and to switch between multicast and unicast mechanisms in a transparent way. Our scheme defines a new protocol, called *httpm*. In addition, it uses currently deployed protocols, and dynamically directs browsers seeking for a URL to multicast channels, while using existing DNS mechanisms. Thus, we enable sites to deliver content to a growing amount of users at less cost and during denial of service attacks, while reducing load on core links. We report simulation results that demonstrate the advantages of the Integrated architecture, and its significant impact on server and network load, as well as clients delay.

1 Introduction

The efficient and timely delivery of web content is one of the most important challenges in today's Internet industry and research community. The need to cut delays while delivering content to a large number of users, along with the high costs of bandwidth at the Internet core, drives the big sites to use caching heavily [1], [2]. However, recent studies suggested that caches are responsible for not more than 50% of the content delivered to users [3], [4]. Content is directly delivered from sites because it is uncachable, dynamically generated, or is not cache-resident for a variety of reasons (further discussion of the above exists in section 2).

In their competition to re-attract users, sites alter content more frequently than before. Information and objects stored on web servers change quite frequently, often every few minutes (e.g., news flashes, bids, stock quotes) [5]. The hotter the news, the bigger is the load on both the site and Internet core links. Frequently, this may result in what the user experiences as either long delays or denial of service. Sites prepare themselves for a maximum peak load by buying more servers, bandwidth and related software and hardware than is needed for the average load. Nevertheless, every (popular) site will eventually cross the planned maximum peak load. Thus, scaling up the number of users while maintaining a sound spending budget can be a real challenge for even the richest of web sites.

In this paper we present an architecture that enables sites to deliver frequently changing information to (an almost) unlimited number of users, in an efficient and scalable manner. Our architecture targets the 2% hottest pages of busy sites, which account for the majority of accesses [6]. At the heart of the architecture is a dynamic distribution selection mechanism that enables the server to identify an increase in demand before the site performance decreases and to activate multicast delivery for high demand pages. When demand abates our mechanism reverts to unicast delivery. We show how this scheme can be integrated into the current web operation mode in a transparent way, and supply mechanisms based on current DNS to dynamically direct browsers seeking URLs to multicast channels. To do so, we define a new protocol specifier, called *httpm*, which replaces the http protocol specifier for the potential hot pages.

Recent works [6], [7] found that not only the access patterns to web sites follow a Zipf-like distribution, but that the alpha parameter, as seen by the server, is larger than previously reported, and is in the range of 1.4 - 1.6. Namely, only a small number of pages at each site (around 2%) account for most of the accesses (over 90%). Moreover, their popularity is stable on time scale of days. (Further discussion on these findings exists in section 2). Our scheme enables browsers to uniquely determine whether the hot page is currently being multicast, and to act upon that. For the deployment of our architecture we suggest two different schemes, which are based on existing building blocks. We use the digital fountain protocol [8] that offers an excellent tradeoff between transmission time and efficiency, for reliable data multicast. Into this protocol we integrate a feedback mechanism based on [9], which enables the site to estimate the size of a hot page multicast group. Both schemes are altered to handle the multicast of dynamically changing content.

The use of multicast for highly popular items has many advantages. It significantly decreases the site load, especially at maximum peak load times, while lowering the amount of traffic in core links. As a result the site is able to serve a larger number of users, even during denial of service attacks. However, multicast itself has a management cost, which consists of join operation and the creation and maintenance of multicast delivery trees. This cost can only be justified when used to save a large amount of individual connections. The dynamic mechanism we propose enables the server to switch between unicast and multicast mechanisms according to the identified load. Our approach is scalable and enables sites to cut down costs, and still be able to handle increased demand. Users benefit from better response time due to lower load on the server, and less congested core links.

An approach to use multicast in the delivery of web resources was first introduced in [10]. In their architecture, clients use http requests. Several simultaneous requests are assembled, and grouped into a multicast group. Then, the clients join the dynamically generated multicast group and receive the page using a reliable multicast protocol. A later work [11] suggested the use of cyclic multicast over UDP for the same architecture, while determining the amount of time the server has to cyclicly transmit the same page until all users in the group receive it. The disadvantage of using this mechanism lies in the unnecessary overhead compared with our scheme: TCP connections are first established, and only then closed. Overhead associated with building and maintaining multicast groups exists for each group, while many groups are created for the same hot page. As a matter of fact, the hotter the page, the more groups are created, and the bigger the overhead.

The integrated architecture model is designed in a way that requires minimal changes to current architectures, and relies mainly on existing mechanisms. Our simulation results show that when switching from the unicast of a hot page to the multicast mechanism, the load on the server decreases significantly, while the both the throughput and the goodput clients experience increases, and may arrive to 4 times th data received in the HTTP model. In the Integrated architecture clients experience a much lower delay than in the HTTP architecture. In addition, the load on core links decreases, thus enabling TCP users to receive better performance.

The rest of the paper is organized as follows. Section 2 gives the motivation for the paper. Section 3 describes the outline of the architecture, while section 4 describes the simulations we conducted and the results obtained from them. In section 5 we describe how this scheme can be implemented as a multi-level scheme, that takes advantage of the current hierarchical structure of the Internet. In section 6 we give our conclusions, and discuss future directions.

2 Motivation

As the Internet is growing, the cost of delivering content to the user at the edge of the Internet has become an important problem. The most intuitive way to decrease costs, is to move content to the edges of the Internet and closer to the user, often to a distributed cache system. The benefits of caching are many. It saves bandwidth, while allowing head-end links to be shared by more clients, to the benefit of the ISPs. It reduces costs to both sites managers and ISPs, by reducing their use of the Internet core links. Caching biggest advantage, which accounts for the rise of companies like Adero, Akamai, Exodus and Sandpiper, is of course the much faster response time a customer experiences, when receiving content from a nearby site [1], [2].

Although caches form a valuable and important solution for improving web performance and decreasing the load from sites servers, a substantial fraction of the content, as seen above, is delivered from the sites themselves, to either the users or the different caches. This content falls into the following categories:

- Uncachable information, which is either personalized information, query results, real time streams, and any time dependent information.
- Dynamically generated data: Several researchers have tried to estimate the update rate of sites. Douglis at al. [5] observed a rather high update rate for about 13% of re-referenced pages, and among the highly referenced html pages, about fifth changed every 15 minutes or less. Overall, they found that a large number of pages are updated periodically. Their findings were also confirmed by Rodrigez and Sibal [12], who found that a large number of web sites update their content randomly, following an exponential distribution. As a result, many potentially cachable resources change fairly rapidly. A recent paper by Padmanabha and Qiu [6], that studied the behaviour of a busy web site, MSNBC, found that server content tends to be highly dynamic, and the duration between successive modifications usually lies between an hour and 24 hours.
- Cache misses and delta encoding deliveries, wherein the site transmits to the different caches either the entire page or only the difference between older and current versions of a page. In order to maintain cache consistency, caches use different Time-To-Live and leasing heuristics [13], [14], [15], [16]. Delta encoding, used in many architectures, takes advantage of the observation that the changes are often minor. It is important to observe, though, that these transmissions also impose some sort of load on the server and core lines, especially for very popular pages, which are cached in many mirror sites.

The question that arises is which type of objects change often, and what is the access pattern to highly changed objects. Breslau at al. [17] determined that page request distribution follows a Zipf-like distribution, but found a weak correlation between access frequency and the rate of change. Douglis at al. [5] found that 16.5% of resources that were accessed at least twice were modified every time they were accessed, and almost half of the text/html resources changed on each access after the first. [6] found that a large part of users requests, and over half of the repeated requests, are to modified files. Access pattern to files follows, as reported in many papers, a Zipf-like distribution [17], [5], [7], [6]. The latters reported that the alpha parameter, as seen by the server, is larger than previously reported, and is in the range of 1.4 - 1.6. This implies, for instance, that just the top 2% of documents account for 90% of the accesses. They also found that the popularity of hot pages tends to be stable over time scale of days.

Our integrated architecture, discussed next, enables the efficient and scalable delivery of dynamically changing content to a large number of users. It's a complementary solution to existing caching mechanisms, which enables also scalable delivery of content to caches.

3 Integrated Architecture Model

The integrated WWW architecture model was designed to allow a smooth transition from TCP based connections to a multicast based mechanism for uncachable hot pages. The model is based on a dynamic selection mechanism, which determines which of the hot pages should be multicast or not, based on inputs received from both the relevant TCP connection requests and UDP feedback information mechanism. We give here an abstraction of the model and the changes required at both the server and client side, and explain how we use existing DNS mechanisms to deliver the information.

3.1 Hot page address resolution

A potential hot page is given in advance a URL with a new protocol specifier, called *httpm*. This assignment is done at the site, for all pages that are candidates for becoming hot ones. As discussed before, the number of such pages is very low, usually less than 2% of the cite's pages. From the client side, multicast web browsing is enabled by a plug-in which recognizes the (*httpm*) prefix. When encountering an *httpm* URL, the plug-in sends the entire URL to the local DNS for address resolution. This is in contrast with usual http requests, where only the server name is sent to the DNS. An *httpm* URL has an independent entry in the DNS, and is accessed by the full URL path. Accessing the DNS with the entire URL requires no change in current DNS mechanisms, since the DNS is a hash table which receives strings and performs a look-up operation. Httpm entries reside on the site's DNS server and will be cached locally only on demand, with SHORT TTL's (longer when the page is multicast). There will be no updates, nor stale records, and a short two-way UDP connection between DNS servers as a result of a DNS query. Once it has received the URL, the DNS then simply returns the IP address in this entry in its query response. In this fashion, our proposed enhancements do not impact DNS software in any way. The web site is responsible for maintaining the correct address for each related *httpm* page in the DNS. This address is either a multicast address, or the site's IP address, in case the page is not currently being multicasted. Upon receiving the DNS response, the client plug-in determines whether the address is a multicast address, or the web server's IP address. If the resolved address is a standard IP address, the client uses the normal HTTP GET mechanism. If the resolved address is a multicast address, the client will seek to join the multicast group. This scheme is scalable the number of HTTPM pages is limited and the URLs are cached in local DNS servers only on demand and for short period. Moreover, since the Zipf distribution is also valid for clients accesses, the number of such cached entries is limited at any specific time at local DNS servers.

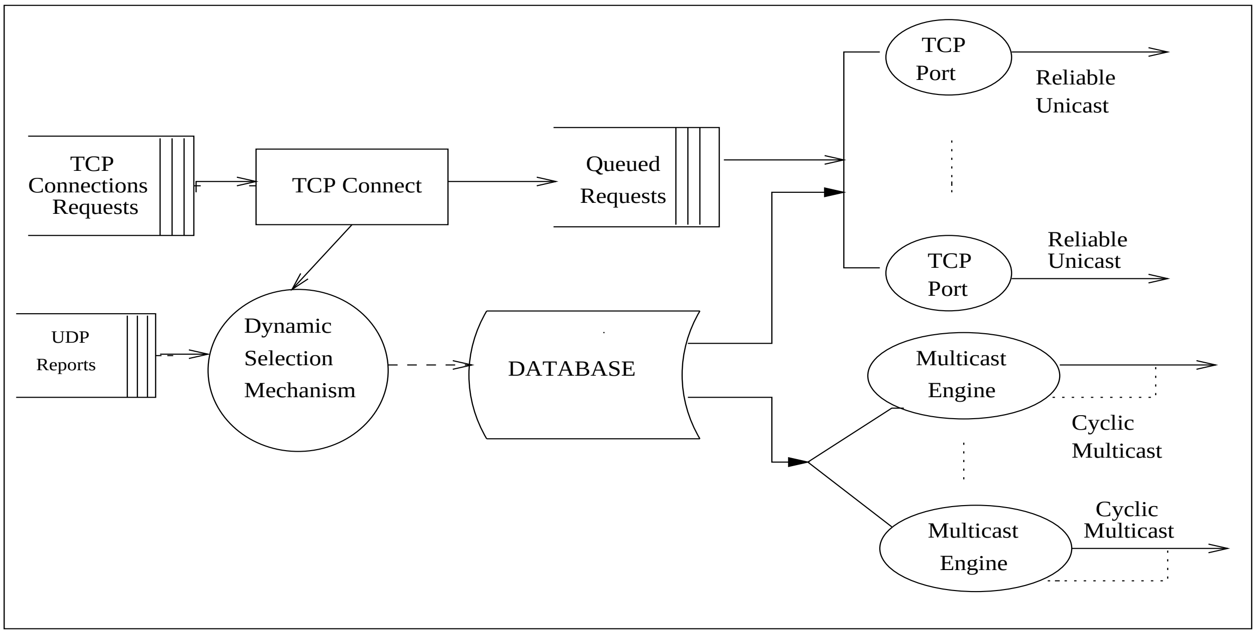

Figure 1: The Site Distribution Selection Mechanism

3.2 Server Side

The site is responsible for maintaining the hot page selection and multicast mechanisms. We classify and motivate here each of its parts. The rest of this section gives a detailed description of each of these parts.

The site predetermines a list of pages which are candidates for becoming hot pages. These pages are given the *httpm* protocol prefix. This list is quasi-static (e.g., Krishnan et al. [7, Fig. 12] showed that the list of popular entities is quite stable, in a medium size site they examined) and thus can be predetermined by the site's administrators off-line. The list can be edited to reflect the relatively slow changes in the popularity of pages, which may demand configuring new entries or deleting existing ones at the DNS.

The dynamic selection mechanism is activated for each of the potential hot pages. It processes the number of incoming connection requests for them. Once a threshold has been crossed for one of the pages, the server begins the process of moving it to the multicast mechanism. It then notifies the DNS of the new address for the appropriate entry.

Once the page is multicast, new connection requests for it are answered with a 3xx HTTP response. This response indicates which multicast address to join, and specifies the needed plug-in. The response has to be determined as part of the HTTP protocol specification.

When the page is multicast, the server estimates every period of time the size of its multicast group. When the estimation crosses down a predefined group size threshold, the site distribution selection mechanism reverts the page from cyclic multicast back to unicast, and notifies the DNS of the new address, the server IP address. While multicast is beneficial for periods of high load, an estimation of the amount of needed resources at other times show it is less beneficial ([18]). Unnecessary edge links are loaded due to the propagation time of prune messages. This unnecessary load at edge networks impacts the performance other clients receive from the network, and therefore should be avoided when possible. It is clear that at high load times, multicast is not worse than concurrent unicast.

Switching a page from multicast to unicast requires a grace period, in which existing members of the group finish receiving the page and new joiners are diverted back to unicast. Therefore, the page is still multicast for a few more rounds, and then a diversion code, which can be recognized by the client browser, is multicast for another period. In this last step the multicast group is exhausted.

3.2.1 Existing Building Blocks

For the multicast mechanism we incorporate two existing building blocks. One is used for efficient delivery, and the other for the group size estimation. Our design describes two different schemes in which to incorporate these building blocks, and enable their use for dynamic content. We give here a short description of these building blocks.

The digital fountain [8] provides an efficient multicast mechanism. Reliability is achieved without the use of feedback messages from clients, and at minimal costs. The digital fountain requires an encoding system at the sending side, i.e., the site, and a decoding mechanism at the receiving side, i.e., the client. In the scheme, a page P_i, which consists of a set of k packets, is encoded and a set of ℓ redundant packets are formed, to a total of n. All n packets are sent cyclically over time. The encoding/decoding process is highly efficient, due to the use of Tornado codes, and therefore imposes very low overhead. The efficiency of the encoding system requires a small stretch factor, which determines the ratio between k and ℓ, even at rather high loss rate times. The digital fountain server design uses layering across multiple multicast groups, and is based mainly on [19]. In this approach, the server organizes the data into g layers, each corresponding to a multicast group. The layers are organized in increasing transmission rate. The layers are cumulative, in that a client who joins layer i, actually joins layers $0..i$. Each layer transmits in an increasing rate. If the ratio between the layers transmission rate is B_i, then a client who joins layer i, receives bandwidth proportional to $2 \cdot B_i$, for $i \geq 1$.

The second building block we use is based on the feedback control mechanism presented by [9], originally intended for multicast video distribution. It requires all participants of a multicast group to generate a random key of 16 bits. The sender sends its key, and awaits an answer from receivers with a matching key. Each period the key is sent with mask of increasing length, until an answer is obtained. Answers also contain the state of the net as perceived by receivers, for rate control mechanisms and timing.

3.2.2 Site Distribution Selection Mechanism

As can be seen in figure 1, the dynamic selection mechanism (DSM) is part of the site's software, which determines whether hot pages are unicast or multicast. The DSM is given the following parameters:

- List of potential hot pages: This list is predetermined by the site's administrators. (A discussion of the quasi-static nature of the list is given above).

- Unicast to multicast threshold: This threshold is actually the number of simultaneous connections that the site cannot tolerate for any one page, and would rather carry the expenses of the multicast delivery. For example, let us assume that it takes an average of 2 seconds to deliver a hot page, when the server is not overloaded and the net is not congested. (For simplicity, we assume the page requires only one connection, as is specified in HTTP1.1). If, during a minute time, the server receives 300 requests for that page, then it has to reserve 10 *simultaneous* connections throughout the minute for the delivery of this page only. We formalize this discussion in the following way:

  Let Δ_i be the average time estimated for any one connection to a page P_i. Let us define the number of requests to P_i in the j th period of time (typically a minute) by R_i^j. Let T_i be the connection threshold, set by the site administrators, for page P_i. Let $S_i^j = R_i^j \cdot \Delta_i$.
  Then, the following conditions should be met, for the DSM to decide to multicast page P_i:

  Threshold
  $$k \in [0...m] : S_i^{j-k} \geq T_i$$
  Monotonically non-decreasing
  $$k \in [0...m-1] : S_i^{j-k} \geq S_i^{j-k-1}$$

  The range $k \in 0..m$ determines the number of periods, in which the conditions above should hold. This interval is needed to determine a pattern, rather than a momentary deviation in requests rate. For example, let $m = 3$. In this case we get that during 3 consecutive periods (minutes) there was a raise in demand for page P_i, and the demand was above the threshold, T_i.

- Multicast to unicast threshold: This threshold is the number of connections that the site can tolerate for a hot page P_i. When this threshold is met during several consecutive estimations done at the DSM, and each time the size of the multicast group decreases, then the DSM switches the page from the cyclic multicast scheme to the HTTP unicast one. As discussed above, this switch is done to decrease potential unnecessary overhead encountered at the edges at low load times.

The DSM obtains the data it needs from two sources: One is the HTTP connections requests, and the other is a counting feedback mechanism. Obtaining the number of requests per hot page is rather straight forward, and involves simple lookup mechanisms. To obtain an estimation of the size of the group, a feedback mechanism based on [9] is activated. This mechanism is part of the multicast mechanism, and will be detailed there.

We assume here that as long as IPv6 is not embedded, multicast addresses constitute a precious resource, that might be rather expensive. Therefore, a site will be able to acquire, either due to the price or due to regulations, only a limited amount of such addresses, and share them between its current hot pages. Thus, even in the absence of high maintenance costs for multicast, a site will be able to use multicast only for a small subset of its pages.

3.2.3 Multicast Mechanisms

We suggest here two different schemes for the multicast of hot pages. Our schemes, which are based on the digital fountain mechanism, also enable an efficient delivery of dynamically changing content. The schemes differ in the levels and rate of transmission, number of concurrent transmission and the way the feedback mechanism is incorporated and used. Each scheme requires different resources from the site, and is intended for different site characteristics. Both schemes cyclicly multicast the data over UDP. Users may join at arbitrary times, and leave once the entire data is obtained.

Since pages change dynamically, a page which is currently transmitted may change on disk. A major requirement in our scheme is to incorporate the change as fast as possible, as users rather have an up-to-date version, if one exists. For this purpose, the header of each multicast packet contains a *continuous* bit. When a page changes on disk, the new version of the page is multicast in the following round, and the bit is toggled. The decoder at the clients side assemblies packets with the same value in the *continuous* bit. If a change is detected, then the decoder discards all packets obtained so far, and starts assembling the packets again, until it can reconstruct the page. Since a change in content occurs only when a new round of the cyclic multicast begins, one bit is sufficient.

Scheme 1: In this scheme the site uses only one channel per page. The site encodes the page for some initial loss rate (for example 10%) and sets the digital fountain's stretch factor accordingly. The encoded information is sent in a rate calculated to match the slowest client possible. Here, the feedback mechanism is used to estimate both the size of the group (for the DSM) and the current loss rate in the net. Since data is transmitted at a low rate, no rate control mechanism is needed.

Once in a few rounds, the feedback mechanism is activated. Every packet in this round contains the following fields:

- The *key* field, which consists of 16 bits, and contains a randomly selected set of 16 bits.
- The *mask* field, which is a byte long, and specifies the number of masks bits in the key. The value of the mask field is determined by the increase in requests rate exhibited at the DSM for this page. Let $R_i^M = \text{Max } R_i^j,\ j \in \tau$. Then, the value of the mask field is $log R_i^M$.

If no answer was received during this round, the mask field is incremented by one, and the new mask field is sent in the next round. The process repeats until answers are obtained.

The client's decoder, upon receiving a packet containing these two fields, randomly picks a key of 16 bits. If the first *mask* digits are the same as in the site's key, it counts the exact number of packets it needs to reconstruct the page, and delivers this information back to the site, along with the read value of the *mask* field.

The site then estimates the size of the group from the number of responses obtained for a round, according to the *mask* field reported. It can also estimate the congestion along the way to the users by the number of packets received until the page could be reconstructed . If this number is bigger than the number of packets sent in a cycle, then the stretch factor is incremented, thus incrementing the amount of encoding information sent.

Scheme 2: This scheme incorporates fully the digital fountain multi-level approach (see section 3.2.1). Therefore, the feedback mechanism does not need to incorporate in it any congestion control information. The mechanism is incorporated as described in scheme 1, and is used only to estimate the size of the group. The two fields query is sent only at the lowest layer, so that it is received by all clients.

3.3 Client Side

Browsers at the client side require an additional plug-in. This plug-in has to identify an *httpm* address, and acquire the address from the DNS. Then, it has to identify whether the address is a server's IP or a multicast address. In the case it is a multicast address, the client joins the multicast group, and activates a decoder, that decodes packets received according to the digital fountain scheme. The plug-in should also identify extra data sent by the counting feedback mechanism and act upon it.

4 Simulation of the Integrated Architecture

In this section we present the results of simulations for the integrated architecture model, compared with simulation results of the current Internet architecture. Our objective is to provide a first order understanding of both the throughput seen by clients and the load on the server at peak times, for both schemes. All results are given for the general case of scheme2. When needed, the private case of scheme1 is also given.

For our simulations we used the *ns-2* ([20]) network simulator. We used a generator that creates Internet-like random topologies with a power law relationship of the node degree (based on [21]). We simulated a WAN network as a network of Autonomous Systems (AS's). Each AS represents either a LAN or a dial-up line switch with active web clients. LAN speed is 10 Mbit per sec, while dial-up line is 56 Kbit per sec. Dial-up line switches have very low connectivity. A web server is located in one of backbone ASs, connected through a dedicated router. It supports a large number of concurrent connections, each is simulated through an *ns* port entity. These ports play the role of pre-forked slave processes in the server, and are either TCP or UDP ports. There are 100 TCP ports and 4 UDP ports, the later transmit in different rates. Arriving TCP connection requests are queued in a *connection request queue*, which simulates the two TCP queues of completed/incomplete connections. It is limited in size, and once filled, in-coming requests are dropped. The unicast mechanism is implemented on top of a two-way full TCP stack, which uses the Reno congestion control protocol. The multicast mechanism we simulated for both schemes. Clients in the AS's make requests to the server, according to a predefined behavior mechanism. The server has 7 available Web pages of different sizes and different popularity. Page size varies according to a normal distribution. The popularity was chosen so that 2 of the pages are responsible for 80% of the accesses. The pages parameters are detailed in the following table:

Type	Size (KBytes)			Popularity
	Minimal	Average	Maximal	
1	25	100	300	4%
2	20	80	240	4%
3	15	50	150	4%
4	10	40	80	4%
5	5	20	60	4%
6	40	40	40	40%
7	60	60	60	40%

We alter the client's wait delay to achieve different loads on the server. Clients may terminate the connection due to slow service either at connection establishment or while receiving the page.

4.1 Goodput

We measured here the amount of data bytes sent by the server vs. the amount of data bytes received by the users for both models. In the Integrated Architecture, the two hot pages (pages 6 and 7) are multicasted at different rates. Figure 2 and Figure 3 show the amount of data sent and received in both architectures. While Figure 2 shows the behavior at peak times, in Figure 3 we gradually increased the load on the server for both architectures. In order to increase the load in the Integrated model, the users request pages at 4 times the rate of users in the HTTP model. The results show, that in the HTTP model, the higher the load, the less received data, up to 30% less received than sent at peak times. In the Integrated model, on the other hand, even at peak periods, the users receive up to 4 times the data sent by the server. As will be seen in this section, not only server load decreases, which enable it to serve TCP clients more efficiently, but also network load decreases. This leads to a much better service to both multicast and unicast users at the Integrated model.

4.2 Server Load

The server load is obtained from several different statistics. First, we examine the ratio between established connections and available out channels (i.e., ports). Obviously, the

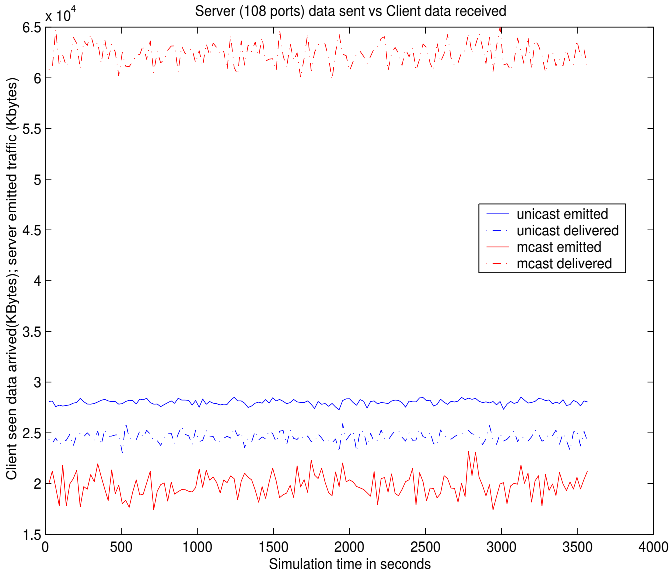

Figure 2: High Load: Transmitted data vs. Received data

server is overloaded when there are awaiting established connections in the queue, and all out channels are busy. Other criteria are the amount of dropped connections from the connection request queue, as well as the rate of establishing connections. Our results show that when the hot pages are multicast, the server load decreases dramatically. Figure 4 shows the amount of load the server maintains. A 100% load is when all the ports are busy, yet the connection request queue is empty. As described earlier, we create more load by shortening the interval between successive client requests. To maintain perspective, we had to quadruple the load in the Integrated model in compare to the HTTP model. Yet, the server reached saturation much earlier (around 1700 seconds) than the Integrated model (around 3000 seconds). Figure 5 demonstrates the server load when we do not quadruple the load in the Integrated model.

The server reaches saturation when all of its ports are busy, and the connection request queue is full. Figure 6 presents the amount of dropped connections from this queue. The HTTP model reaches saturation much quicker than the Integrated model, and when it does, the amount of dropped connections is much higher.

4.3 Clients Seen Delay: Server vs. Network load

We measured the delay perceived by clients in both architectures as a function of time, while increasing the load with time until we saturate the server (see Figure 4).

Figure 7 shows the amount of time it takes users in both architectures to receive pages 6 and 7, the hot pages that are multicatsed in the integrated architecture. The results show quite clearly that clients in the Integrated model receive these pages much faster than in the HTTP model. Furthermore, it is beneficial to multicast these pages even if the server is not

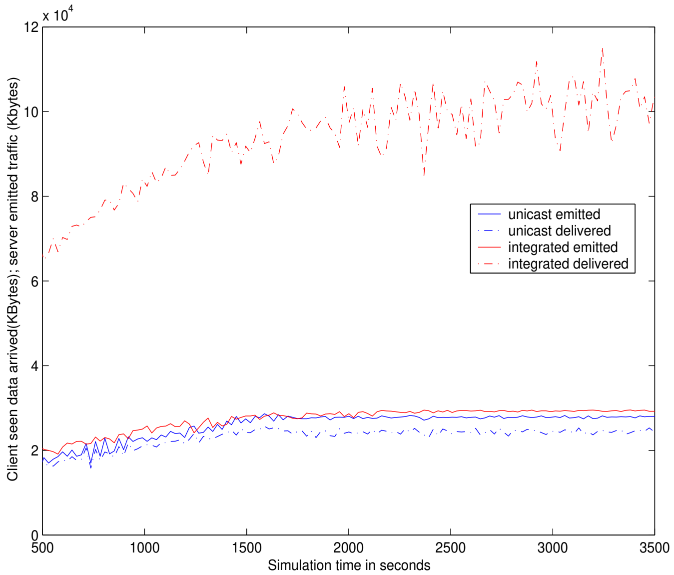

Figure 3: Increased Load: Transmitted data vs. Received data

loaded (we refer here to scheme 2). As the load increases, our results show that the delay a user suffers in the HTTP model can be 4 times the delay in the Integrated model.

Figure 8 shows the amount of time it takes clients to receive pages which are unicast in both models. Page 1 represents a large file, Page 3 a medium size file and page 5 a rather small file. All three files suffer smaller delays on their way to the user in the Integrated model, and the effect is more obvious with the increase in server load. A similar effect can be seen when viewing the first packet delay, in Figure 9. As the load on the server increases, it takes longer to create new connections in both architectures, although significantly longer for the HTTP model.

An interesting point is that the smaller the page, the bigger is the difference in delay between the two architectures. Although this can also be explained by first packet delay, which is more significant for shorter connections, we suspected there is another factor that influences this delay - the load on the network itself. Figure 10 shows the percentage of retransmitted packets from the server, due to time outs in the TCP protocol. These time outs indicate that the network is congested, which causes the TCP flow control mechanism to adjust its window size and slow down the connection. This retransmission percentage, although quite low in both models, is more than double the size in the HTTP model than in the Integrated model, indicating that the network is more congested in the HTTP model. Figure 11 shows the amount of connections terminated by users, due to too long delays. As soon as the server reaches peak load, clients start to give up on connections. Since the server in our simulation does not impose additional delays on established connections, then this extra delay is due to the TCP flow control mechanism, that adapts to a congested network.

From the above we conclude, that one of the reasons for the degradation in performance seen by clients at peak times is the congestion in core links. The closer the link is to the

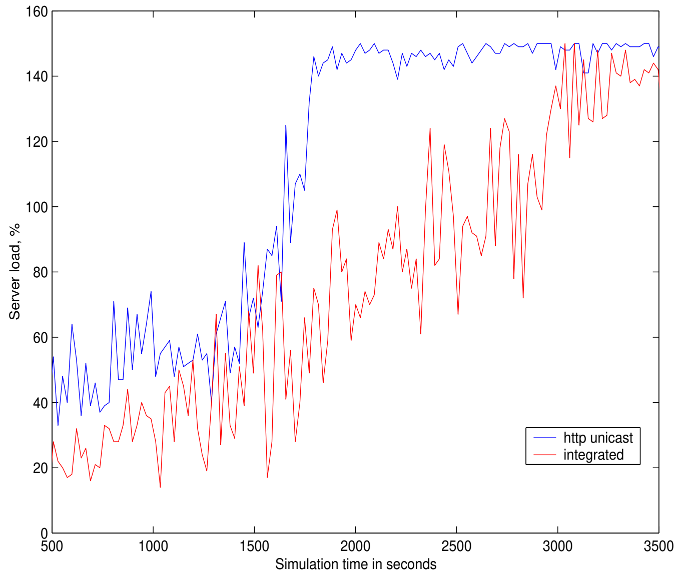

Figure 4: Server Load

server, the more congested it can become. Although the TCP congestion control mechanism tries to limit the effect of such peak times, by exhibiting a social behavior, the effect on both the site and the client is big. The site retransmits an extensive amount of packets until the TCP slow start mechanism takes effect, while the client experiences degradation in performance. The use of multicast mechanism for the delivery of hot pages, reduces both the traffic on core links, and the load on the server. Clients which connect by http at such times receive better service, and the amount of retransmission decreases significantly.

4.4 Special Case: Scheme1

We look here at the special case, where the server uses only one outgoing channel per multicasted page, therefore transmitting it in the rate of the slowest receiver. In our simulation, it means at a rate of 56Kbits per second.

Figures 12, 13 and 14 show that the load on the server decreases significantly in the Integrated model, while the goodput is much higher than in the HTTP model.

The special case of scheme 1 enables us to investigate the effect of using multicast instead of unicast on the network itself. Since the server transmits UDP data in a low rate, clients receive hot pages rather slow, and the server cannot be over loaded in the Integrated architecture. Therefore, when we quadruple the amount of requests made by clients in the Integrated architecture, we create extra load on the network, rather then on the server. Figures 15 and 16 show the effect of slow UDP connection on congested network - fast receivers are bound to lower rate when receiving multicast packets, while TCP packets are delivered in close to dull speed. As traffic increases, UDP packets that are lost cause receivers to wait till the next multicast cycle. Multicasting hot pages in low rates is therefore beneficial only

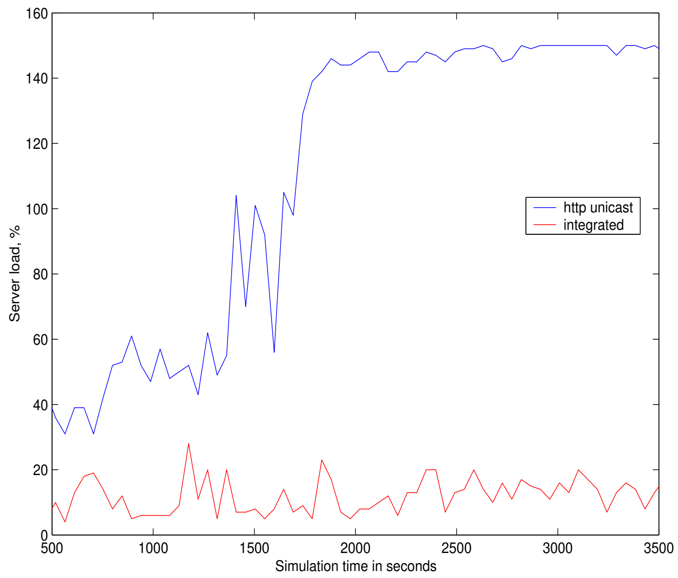

Figure 5: Server Load - Equal Delays between Clients Requests

when the server is overloaded.

Figures 17 and 18 show that HTTP clients receive better performance when the server is not loaded, and therefore the Integrated model is indeed beneficial also in scheme 1, when the server becomes loaded.

5 Multi Level Architecture

In this section we present a way to integrate our mechanisms into the current hierarchical structure of the Internet. While our integrated architecture was first designed for sites, it seems just as beneficial for proxies, in general, and for proxies used by Internet service providers, in particular.

Current ISPs maintain big autonomous networks and support very large number of users. In the integrated architecture suggested so far proxies play the role of clients. However, for their clients, proxies may take the role of the server. Thus, we suggest here the implementation of our integrated architecture one (or more) levels deeper. In this implementation, once a proxy detects an increase in the number of requests to a specific page, it multicasts this page to its clients, using our scheme. Using multicast for the delivery of hot pages within the ISPs' subnetwork can be very beneficial not only to the users but to the ISP as it both reduces its internal traffic and give its paying customers better service. It has special perfomance benefits when the structure of the subnet is tree-based, such as with cable network based service providers.

While the implementation of the scheme in the ISPs level is straight forward for hot pages with the *httpm* protocol specifier, another scheme is needed for locally cached hot

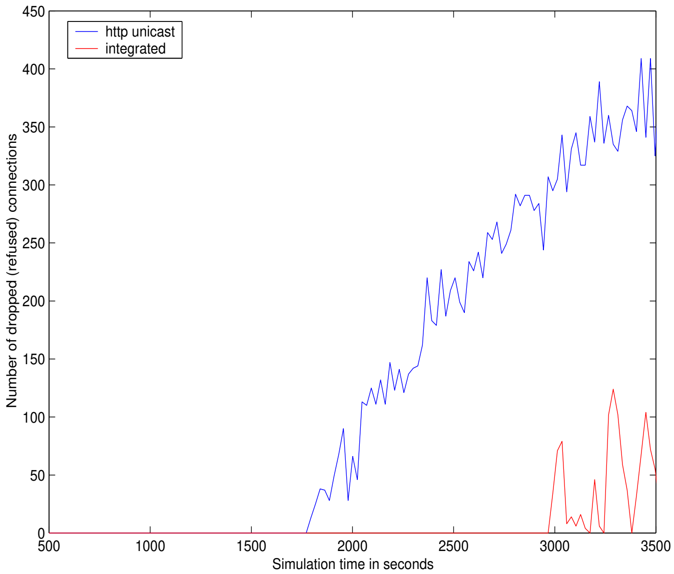

Figure 6: Dropped Connections

pages. Such a scheme is beyond the scope of this paper, and requires further research.

6 Conclusions

Frequently changing web content requires the use of new mechanisms for its scalable delivery. Our integrated architecture enables the efficient and scalable delivery of such content. Its main advantage is its simplicity and transparency to users. Our results show that it allows sites to serve a growing amount of users at times of peak in load without experiencing performance degradation. Users delay decreases significantly, and both goodput and throughput quadruple. An important conclusion from our simulations is that, many times, performance degradation is due to increased load on core links. This increase is caused from the large amount of concurrent connections to sites, all aimed to get the exact same information. The result is congested links, which have the immediate effect of increasing the load on both the sites and the links in the short run because of retransmissions. Our scheme solves this problem by using multicast at such times, thus enabling economical use of core links.

Further research and measurements of the impact of peak times on the load of both sites and core links is required. In addition, our model serves as a good test bed for the behavior of single source multicast, when implementing the different schemes. While this paper proves the feasibility and advantages of our scheme, the incorporation of this scheme in the Internet is challenging. Further examination of its impact on both core and edge links, once implemented, can quantify the trade-offs between the unicast and the multicast model. This is required in order to set better ground rules as to when to use which.

Acknowledgments

The authors would like to thank Dahlia Malkhi for her suggestions and helpful remarks. We would also like to thank Ohad Rodeh for his help in building the simulated system.

References

[1] A. Dornan, "Farming out the web servers," *NPN: New Public Network Magazine*, vol. http://www.networkmagazine.com, Mar. 2000.

[2] J. Angel, "Caching in with content delivery," *NPN: New Public Network Magazine*, vol. http://www.networkmagazine.com, Mar. 2000.

[3] J.C. Mogul, "Squeezing more bits out of http caches," *IEEE Network mamgazine*, pp. 6 – 14, May 2000.

[4] A. Wolman, G.M. Voelker, N. Sharma, N. Cardwell, A. Karin, and H.M. Levy, "On the scale and performance of cooperative web proxy caching," *In Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP99)*, pp. 16 – 31, Dec. 1999.

[5] F. Douglis, A. Feldman, B. Krishnamurthy, and J. Mogul, "Rate of change and other metrics: a live study of the world wide web," *In Proceedings of USENIX symp. on Internet Tech. and Systems Monterey, CA*, Dec. 1997.

[6] V.N. Padmanabhan and L. Qiu, "The content and access dynamics of a busy web site: findings and implications," *ACM SIGCOM*, Aug 2000.

[7] P Krishnan, Danny Raz, and Yuval Shavitt, "The cache location problem," *IEEE/ACM Transactions on Networking*, vol. 8, no. 5, Oct. 2000, to be published.

[8] J.W. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data," *Sigcomm'98, Vancouver, Canada*, Sept. 1998.

[9] J.C. Bolot, T. Turletti, and I. Wakeman, "Scalable feedback control for multicast video distribution in the internet," *ACM Sigcomm'94, London, UK*, pp. 58 – 67, Sept. 1994.

[10] R. Clark and M. Ammar, "Providing scalable web services using multicast communication," *Proceedings of the IEEE Workshop on Services in Distributed and Networked Environments, Whistler, Canada*, Feb. 1995.

[11] Z. Fei K.C. Almeroth, M. Ammar, "Scalable delivery of web pages using cyclic-best-effor (udp) multicast," *Infocom'98*, Mar. 1998.

[12] P. Rodriguez and S. Sibal, "Spread: Scalable platform for reliable and efficient automated distribution," *WWW9*, May 2000.

[13] C. Gray and D. Cheriton, "Leases: An efficient fault tolerant mechanism for distributed file cache consistency," *In Proceedings of the 12th ACM Symposium on Operating Systems Principles*, pp. 202–210, 1989.

[14] D. Wessels, "Squid internet object cache. http://www.nlanr.net/squid/," 1996.

[15] C. Liu and P. Cao, "Maintaining strong cache consistency in the world wide web," *In Proceedings of the ICDCS*, May 1997.

[16] V. Duvvuri, P. Shenoy, and R. Tewary, "Adaptive leases: A strong consistency mechanism for the word wide web," *In Proceedings of Infocom'00*, vol. 2, pp. 834 – 843, Mar. 2000.

[17] L. Breslau, P. Cao, L. Fan, G. Philips, and S.Shenker, "Web caching and zipf-like distributions: evidence and implications," *Infocom'99*, Mar. 1999.

[18] K. Almeroth and M. Ammar, "Multicast group behavior in the internet's multicast backbone (mbone)," *IEEE Communications*, June 1997.

[19] L. Vicisano, L. Rizzo, and J. Crowcroft, "Tcp-like congestion control for layered multicast data transfer," in *Infocom'98*, Mar. 1998.

[20] "UCB/LBNL/VINT Network Simulator - ns (version 2), 1997," URL: http://www.isi.edu/nsnam/ns.

[21] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos, "On power-law relationships of the internet topology," in *ACM SIGCOMM*, Aug. 1999.

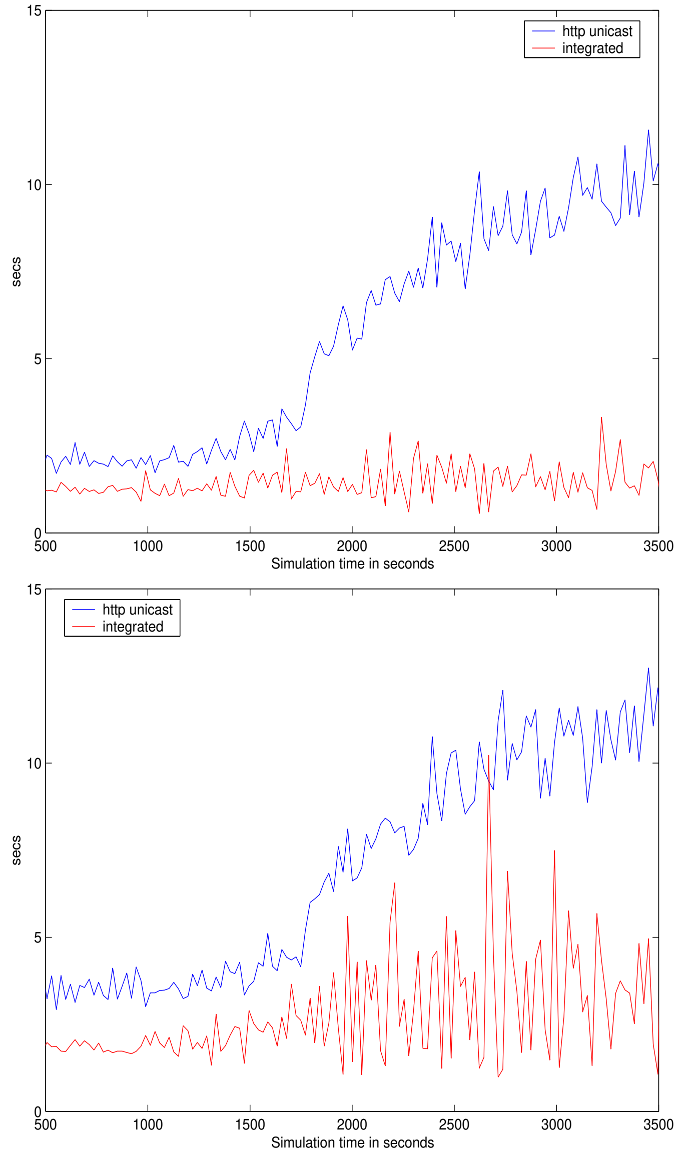

Figure 7: Client Seen Delay: pages 6 and 7

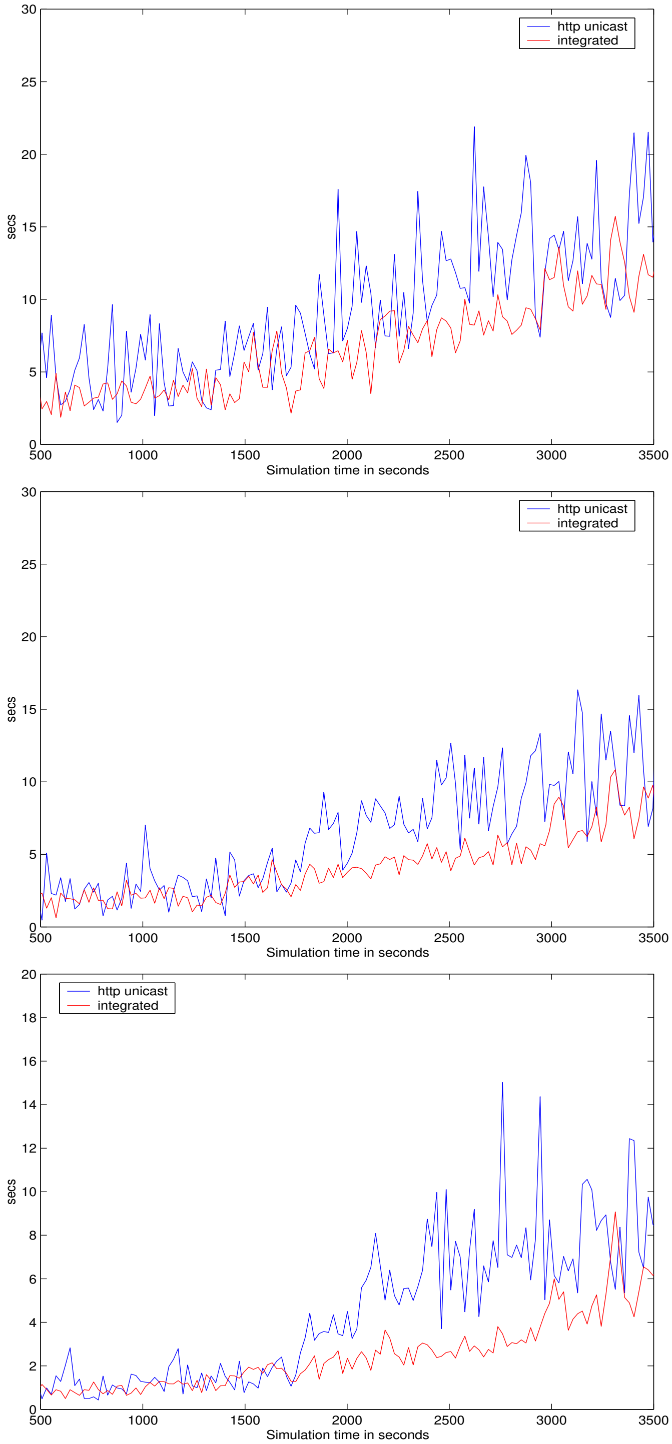

Figure 8: Client Seen Delay: pages 1, 3 and 5

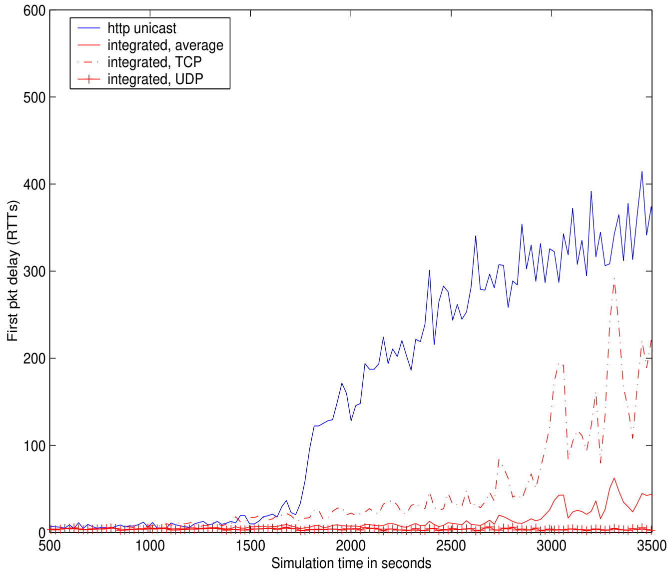

Figure 9: First Packet Delay

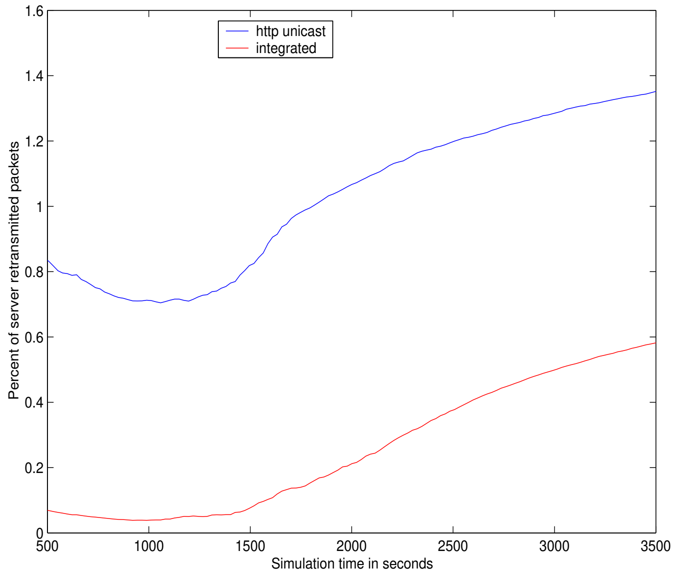

Figure 10: Retransmissions from the Server

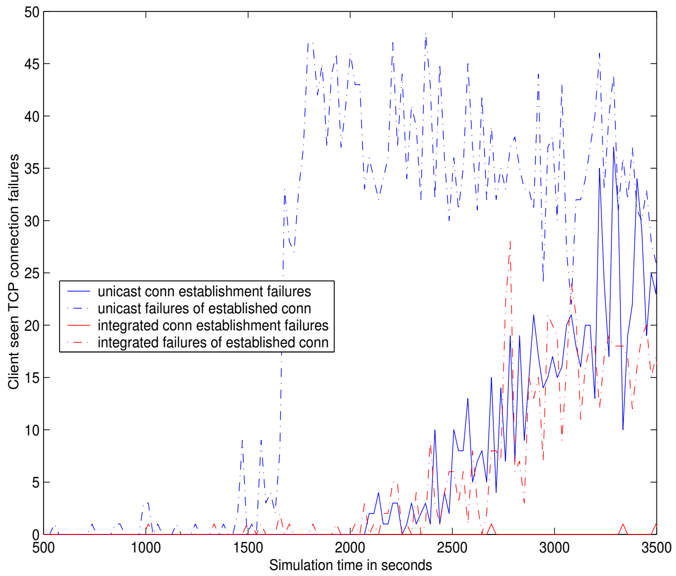

Figure 11: Established Connections Failures

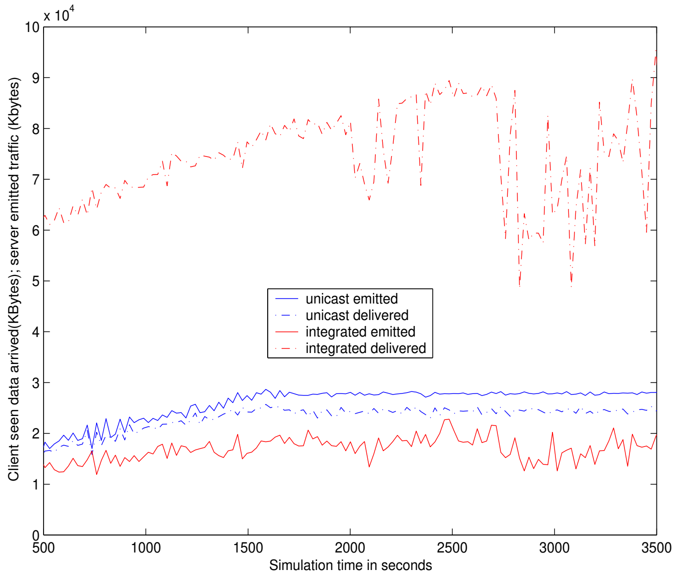

Figure 12: Scheme1 Increased Load: Transmitted data vs. Received data

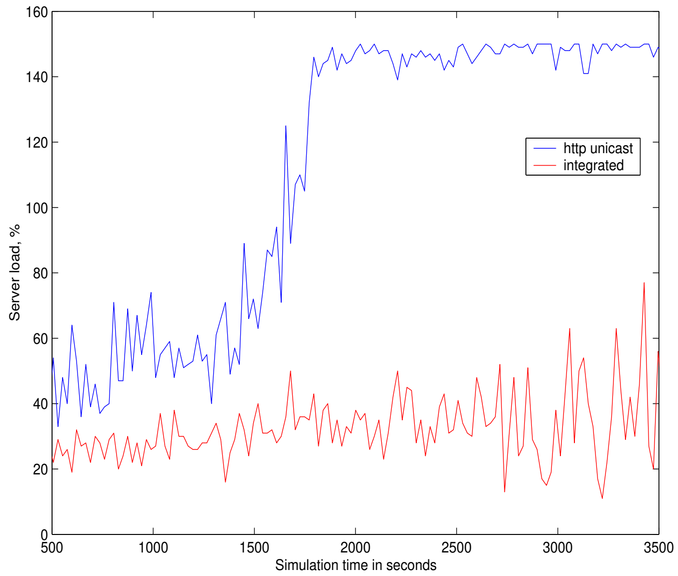

Figure 13: Scheme1: Server Load

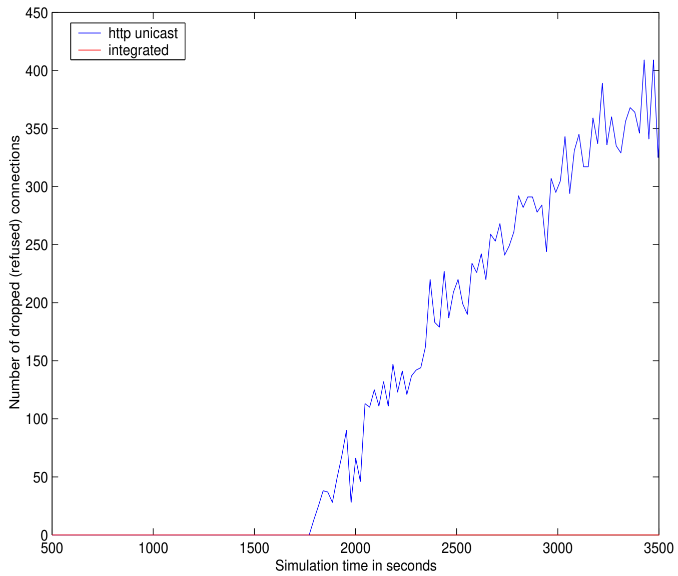

Figure 14: Scheme1: Dropped Connections

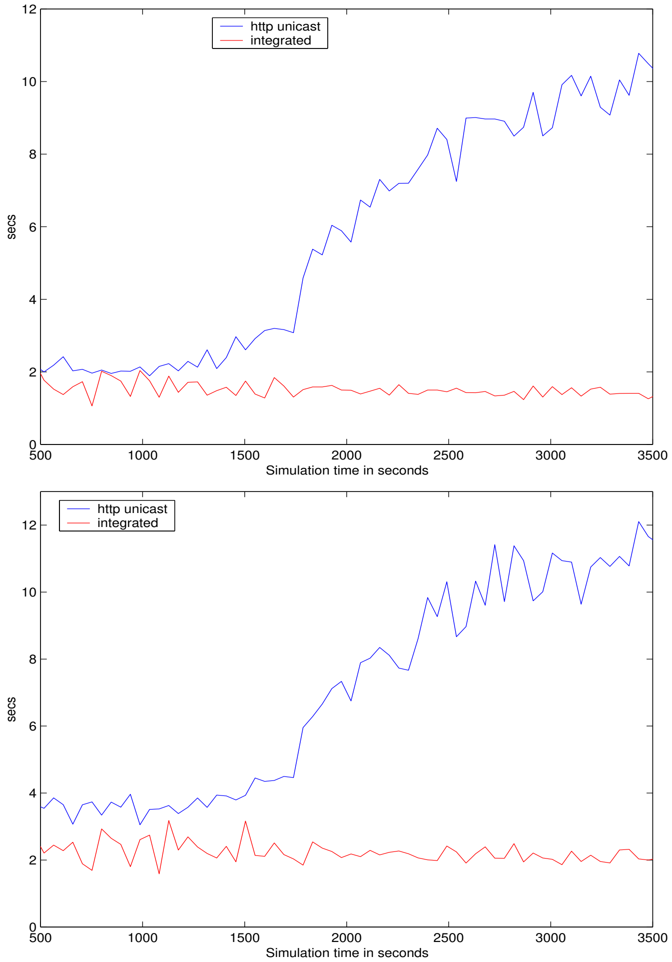

Figure 15: Scheme1 Client Seen Delay: pages 6 and 7

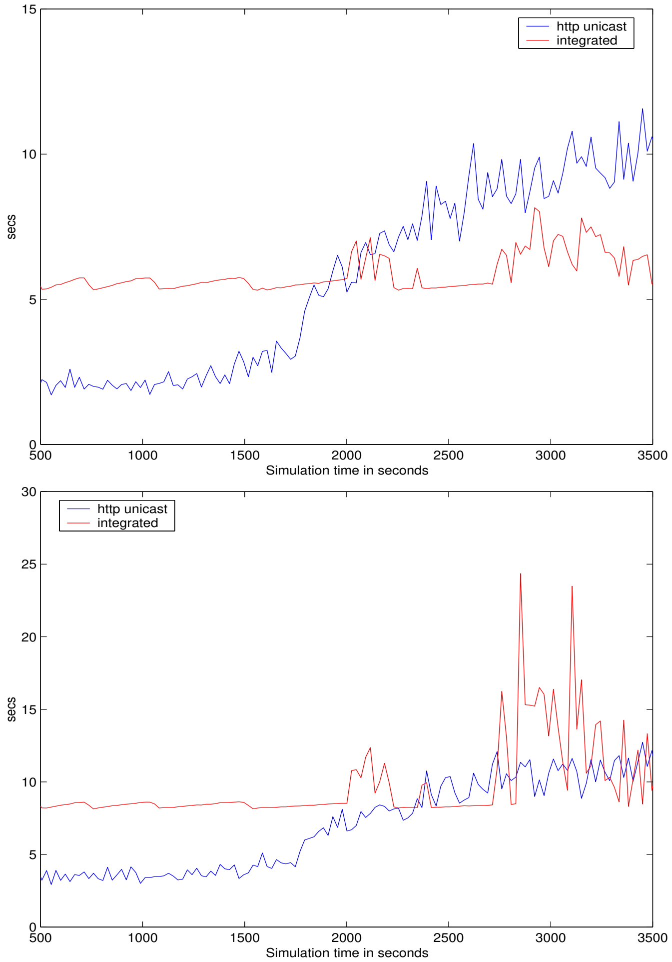

Figure 16: Scheme1, quadruple integrated request rate, Client Seen Delay: pages 6 and 7

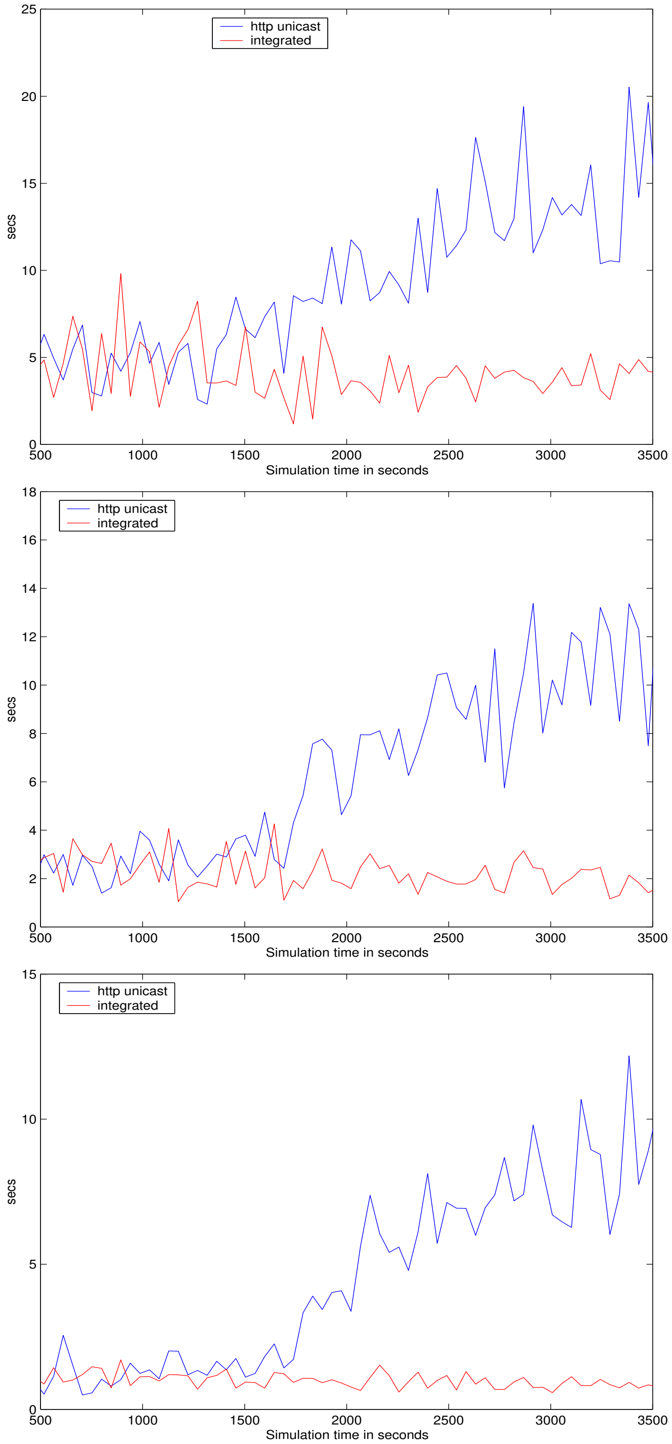

Figure 17: Scheme1, quadruple integrated request rate, Client Seen Delay: pages 1, 3 and 5

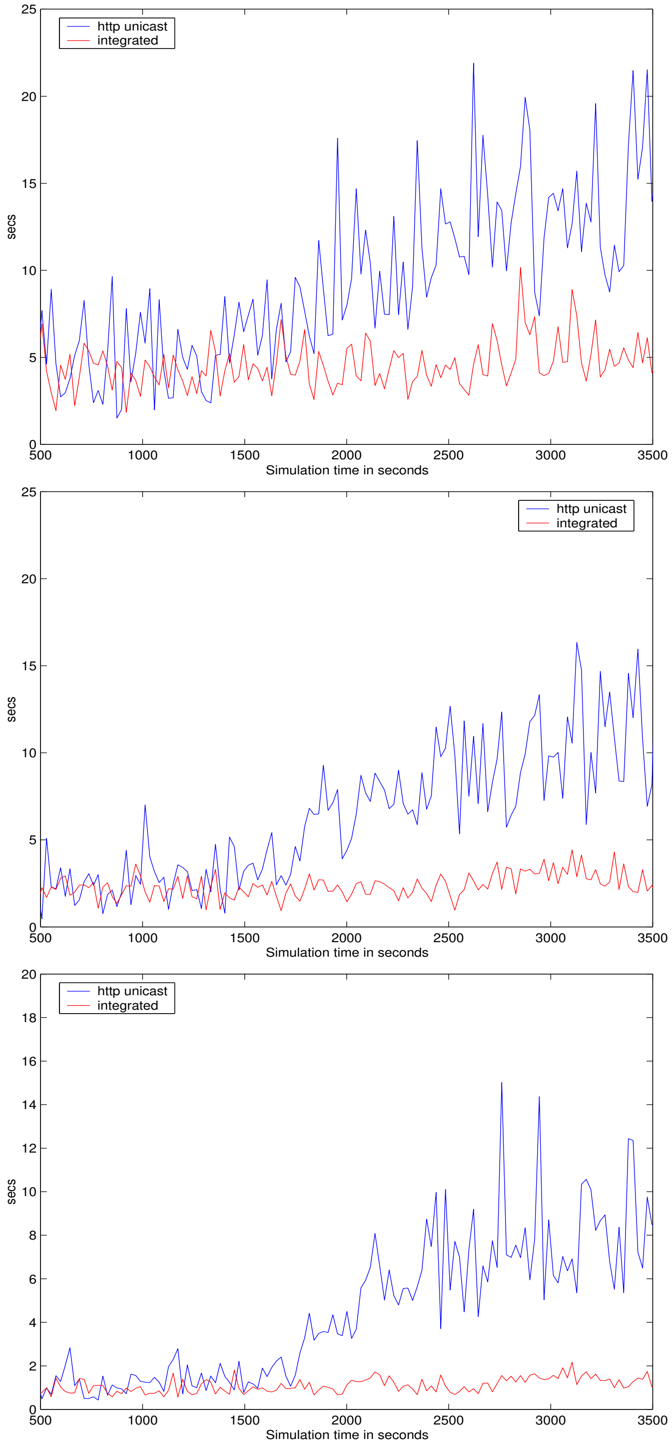

Figure 18: Scheme1 Client Seen Delay: pages 1, 3 and 5