Understanding the Management of Client Perceived Response Time

David Olshefski
IBM T.J. Watson Research Center
19 Skyline Drive
Hawthorne, NY 10532
olshef@us.ibm.com

Jason Nieh
Department of Computer Science
Columbia University
1214 Amsterdam Avenue MC0401
New York, NY 10027
nieh@cs.columbia.edu

ABSTRACT

Understanding and managing the response time of web services is of key importance as dependence on the World Wide Web continues to grow. We present *Remote Latency-based Management* (RLM), a novel server-side approach for managing pageview response times as perceived by remote clients, in real-time. RLM passively monitors server-side network traffic, accurately tracks the progress of page downloads and their response times in real-time, and dynamically adapts connection setup behavior and web page content as needed to meet response time goals. To manage client perceived pageview response times, RLM builds a novel event node model to guide the use of several techniques for manipulating the packet traffic in and out of a web server complex, including fast SYN and SYN/ACK retransmission, and embedded object removal and rewrite. RLM operates as a stand-alone appliance that simply sits in front of a web server complex, without any changes to existing web clients, servers, or applications. We have implemented RLM on an inexpensive, commodity, Linux-based PC and present experimental results that demonstrate its effectiveness in managing client perceived pageview response times on transactional e-commerce web workloads.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling Techniques; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Client/server*

General Terms

Management, Measurement, Performance, Experimentation.

Keywords

Web server performance, client perceived response time, QoS, admission control.

SIGMetrics/Performance '06, June 26-30, 2006, Saint Malo, France.

1. INTRODUCTION

Response time is a key indicator of end user satisfaction in using web services. Customers seeking quality online services have choices, and will simply take their business elsewhere when response times exceed acceptable thresholds. As dependence on the World Wide Web continues to grow, it is increasingly important for businesses to understand and manage the response times experienced by their clients.

Although many techniques have been developed for managing the response time of web services [1, 3, 8, 9, 12, 13, 16, 22, 36], previous approaches focused on controlling only the web server response time of individual URL requests. Unfortunately, this has little relevance to end users who are located remotely, not at the web server, and who are interested in viewing entire web pages that consist of multiple objects, not just individual URLs. The problem is exacerbated by the fact that the response time measured within the web server can be an order of magnitude less than that perceived by the remote client [26]. These techniques are controlling the wrong measure of response time. They may in fact improve server response time while unknowingly and unexpectedly degrade the overall response time seen from the perspective of end users [26].

Managing the response time of web services is crucial for providing differentiated services in which different classes of traffic or clients can receive different quality of service. For example, an e-commerce web site may want to ensure that users with full shopping carts are given highest priority for receiving the best response time while users who are casual visitors are given lower priority. Existing approaches that provide differentiated services often depend on load shedding in the form of admission control to maintain a specified set of response time thresholds [9, 16, 12, 22]. Requests from low priority clients are dropped when they begin to interfere with the response time of high priority clients. However, prior techniques ignore the effect that admission control drops have on the overall response time perceived by end users.

We present *Remote Latency-based Management* (RLM), a novel approach for managing web response times as perceived by remote clients using only server-side techniques. RLM correlates container pages and their embedded objects to manage pageview response times of entire web pages, not just individual URLs. RLM tracks the progress of each web page in real-time as it is downloaded, and uses the information to dynamically control the client perceived response time by manipulating the network traffic in and out of the

web server complex. RLM provides its management functionality in a non-invasive manner as a stand-alone appliance that simply sits in front of a web server complex, without any changes to existing web clients, servers, or applications.

RLM builds on our previous work on ksniffer [27], a kernel-based traffic monitor that accurately estimates pageview response times as perceived by remote clients at gigabit traffic rates. RLM uses passive packet capture to track the elapsed time of each pageview download, then uses this information to build a novel event node model to enable RLM to make key management decisions dynamically at each point in the download process. In particular, our model defines and includes the effect of connection admission control drops on partially successful web page downloads. It also accounts for some notable behaviors of common web browsers in the presence of connection failures.

Using this model, RLM applies two sets of techniques for managing pageview response time, fast SYN and SYN/ACK retransmission, and embedded object removal and rewrite. Fast SYN and SYN/ACK retransmission reduce connection latencies associated with bursty loads and network loss, which are key factors for short-lived TCP connections typical of web transactions. Embedded object removal and rewrite reduce server and network transfer latencies by adapting web page content in terms of how embedded objects are handled depending on the elapsed pageview response time at the point at which objects are requested. These techniques can be applied using a simple set of management rules that can be defined to provide differentiated services across multiple classes of web clients and content.

We implemented RLM on an inexpensive, commodity, Linux-based PC and demonstrated that it can manage client perceived pageview response times in real-time for three-tier web architectures. Using our prototype, we present some experimental data obtained from using RLM for managing client perceived pageview response times using the TPC-W e-commerce web workload. We present results for both single and multiple service class environments. Our results show RLM's unique ability to track a pageview download as it occurs, properly measure its elapsed response time as perceived by the remote client, decide if action ought to be taken at key junctures during the download, and apply latency control mechanisms for the current activities.

This paper introduces RLM as a first step toward managing client perceived response times for web transactions. Section 2 presents an overview of the RLM architecture. Section 3 describes the RLM pageview download model and pageview event node framework used for making response time management decisions. Section 4 describes RLM mechanisms used for connection latency management and their effect on client browsers. Section 5 describes RLM mechanisms used for page transfer latency management. Section 6 describes the implementation of RLM and presents some experimental results based on managing a TPC-W workload. Section 7 discusses related work. Finally, we present some concluding remarks and directions for future work.

2. RLM ARCHITECTURE OVERVIEW

As shown in Figure 1, RLM is a stand-alone appliance which sits in front of a web server complex. RLM does not require any modifications to Web pages, the server complex, or browsers, making deployment fast and easy. This is particularly important for Web hosting companies that are

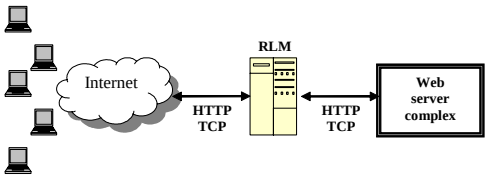

Figure 1: RLM deployment.

responsible for maintaining the infrastructure surrounding a Web site, but are not permitted to modify the customer's server machines or content.

RLM builds on our previous work on ksniffer [27] and uses a similar architecture. It is designed as a set of dynamically loadable kernel modules that reside above the network device independent layer in the operating system. Its device independence makes it easy to deploy on any inexpensive, commodity PC without special NIC hardware or device driver modifications. RLM monitors and manages bidirectional network traffic and looks at each packet only once. Its in-kernel implementation exploits several performance optimizations such as as zero-copy buffer management, eliminated system calls, and reduced context switches [18, 21]. Our previous work demonstrated that the in-kernel architecture can support gigabit traffic rates [27].

RLM operates as a server-side mechanism with a low-delay control path to the web server complex that is unaffected by outside network conditions. RLM measures client perceived pageview response times for web transactions, then uses that information as real-time feedback in managing the behavior of the web server complex to deliver desired response times. This tight measurement and management feedback loop near the server complex is key to RLM's ability to provide real-time control of the performance of the web server complex.

RLM passively captures network packets to measure client perceived response times, then actively manipulates the packet stream between client and server to meet desired response time goals. RLM operates at the network packet level in part to provide its functionality without any modifications to the web server complex. More importantly, as discussed in Section 3, measuring and controlling client perceived response times require tracking the client-server interaction at the packet level.

RLM measures client perceived response times by capturing and analyzing packets using a model of TCP retransmission and exponential backoff that accounts for latency due to connection setup overhead and network packet loss. It combines this model with higher level online mechanisms that use access history and HTTP referrer information when available to learn relationships among Web objects to correlate connections and web objects to determine pageview response times. Our previous work presents further detail on how client perceived response time measurements are performed [27]. The remainder of this paper focuses on the management mechanisms RLM provides that work in conjunction with its measurement mechanisms.

3. RLM PAGEVIEW EVENT NODE MODEL

RLM introduces a new model for specifying and achieving response time service level objectives based on tracking a pageview download as it happens and making service decisions at each key juncture based on the current state of the

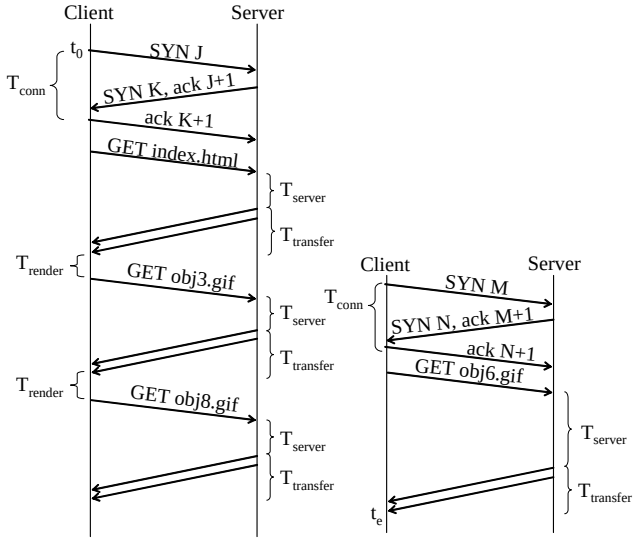

Figure 2: Breakdown of client response time.

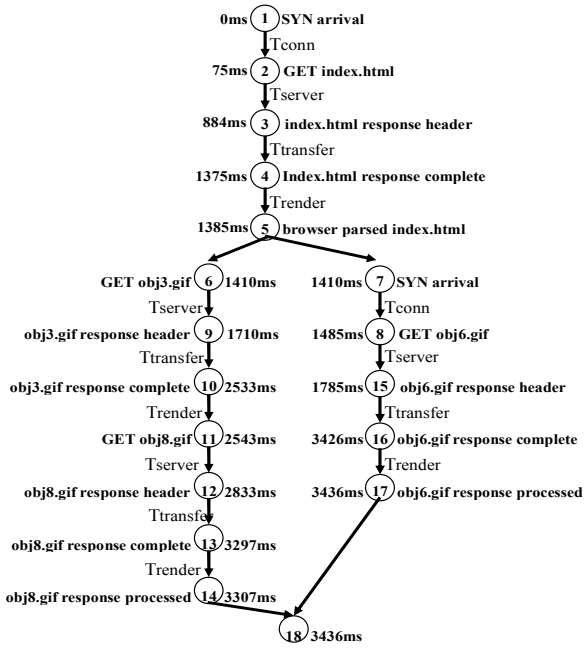

Figure 3: Pageview modeled as an event node graph.

pageview download. A pageview download can be viewed as a set of well defined activities such as establishing a connection, getting the container page, and getting the embedded objects in a page. RLM models a pageview download as an event node graph, where each node represents an activity and each link indicates a precedence relationship. The nodes in the graph are ordered by time and each node is annotated with the elapsed time from the start of the transaction. Each activity contributes to the overall response time. Some activities may overlap in time, have greater potential to incur larger latencies, be on the critical path, and be more difficult to control than others. RLM controls pageview response times by identifying and managing the high latency activities on the critical path of the pageview download.

To illustrate our approach, Figure 2 depicts the response time of $t_e - t_0$ for the pageview download of *index.html* which embeds *obj3.gif*, *obj6.gif* and *obj8.gif*. The example uses two connections to download the page, consistent with modern web browsers which open multiple connections to download web content faster. Four types of latencies are serialized over each connection and delimited by specific events:

1. T_{conn} TCP connection establishment latency, using the TCP 3-way handshake. Begins when the client sends the TCP SYN packet to the server.

2. T_{server} latency for the server complex to compose the response by opening a file, or calling a CGI program or servlet. Begins when the server receives an HTTP request from the client.

3. $T_{transfer}$ time required to transfer the response from the server to the client. Begins when the server sends the HTTP request header to the client.

4. T_{render} time required for the browser to process the response, such as parse the HTML or render the image. Begins when the client receives the last byte of the HTTP response.

The corresponding event node graph generated by RLM is shown in Figure 3. Each link is identified with the type of latency that results from the particular activity. Each node is annotated with the elapsed time from the start of the transaction. By measuring the elapsed time at a given node, RLM can track the page download as it progresses and determine at each node whether to take additional actions to satisfy response time goals for the given page. This ability to make management decisions at each point in time within the context of the pageview download is a key difference between RLM and other QoS approaches.

It is crucial for RLM to model network loss and track client-server interaction at the packet level to measure and manage the entire client perceived response time $t_e - t_0$ shown in Figure 2. Mechanisms which attempt to measure response time via time stamping server-side user space events are ineffective. For example, measuring response time within an Apache web server ignores time spent during the TCP 3-way handshake for establishing the connection and time spent in kernel queues before the request is given to Apache. Such measurements have shown to be an order of magnitude less than the response time experienced by the remote client [26]. Likewise, measuring and controlling the time required to service a single URL (i.e. T_{server}) is simply not relevant to the remote client who is downloading not just a single URL but an entire pageview. Latency control mechanisms must take into account effects seen at the packet level which impose latency for the remote client.

RLM allows a variety of rules to be defined and enforced at different points in an event node graph to manage response time. Each node in the graph has a set of associated characteristics that determine what types of rules can be defined. For example, when a SYN is captured by RLM at node 1 in Figure 3, the management decision can be based only on fields contained within the SYN packet, namely source IP address, destination IP address, source port, destination port. The decision cannot be based on which page is being requested since the GET request has not yet been sent by the client. Algorithms for quickly classifying packets or clients into service classes have been previously studied and are not

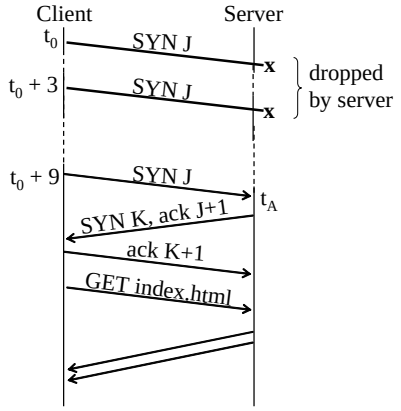

Figure 4: SYN drops at the server.

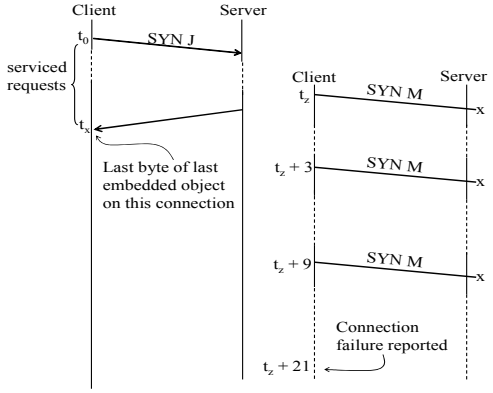

Figure 5: Second connection in page download fails.

discussed in this paper due to space constraints. We focus instead on the management framework and on introducing a set of techniques that can be applied for each instance of a page download to reduce the remaining time to complete the pageview to meet a defined response time goal. Section 6 provides some example rules that can be used with RLM and their impact on response time.

4. CONNECTION LATENCY MANAGEMENT

One of the key types of latency that RLM must manage is TCP connection establishment latency T_{conn}. It is especially important to understand its impact on client perceived pageview response times since a great deal of work in controlling web server performance has focused on applying admission control to prevent web server overload by dropping TCP connections. However, the effect of admission control drops on the behavior of web browsers has not been carefully studied.

To determine how load shedding affects the client perceived response time on real web browsers, we conducted a series of experiments using Microsoft Internet Explorer v6.0 and Firefox v1.02 in which we performed various types of connection rejection by performing SYN drops to emulate an admission control mechanism at the web server. The end result was that the resulting response time at the browser is greatly affected not only by the number of SYN drops, but also by the connection for which the SYN drops occur.

Figure 4 depicts the behavior of TCP under server SYN drops. The client sends the initial SYN at t_0, but the server drops this connection request due to admission control. The client's TCP implementation waits 3 seconds for a response. If no response is received, the client will retransmit the SYN at $t_0 + 3s$. If that SYN gets dropped, then the next SYN retransmission occurs at time $t_0 + 9s$. The timeout period doubles (from 3s, 6s, 12s, etc.) until either the connection is established, the client hits stop/refresh on the browser which cancels the connection, or the maximum number of SYN retries is reached. This is the well-known TCP exponential backoff mechanism.

Server SYN drops are not a denial of service, but rather a means for rescheduling the connection into the near future. Although this behavior is effective in shedding server load, it has significant effects on the response time perceived at the clients. Existing admission control mechanisms which perform SYN throttling simply ignore this effect and report the response time once the connection is accepted, beginning from time t_A. This misrepresents both the client perceived response time and throttling rate at the web site.

Because web browsers open multiple connections to the server as shown in Figure 2, it is important to understand the effect of a SYN drop in the context of which connection is being affected. If only the first SYN on the first connection is dropped, then the client will experience the 3s retransmission delay, but will still be serviced. If the first connection gets established immediately, but all SYNs on the second connection are dropped as shown in Figure 5, then the client will eventually receive a connection failure after multiple retries. While the second connection is undergoing SYN drops at the server, our study shows that web browsers will display an hourglass cursor on the screen, a spinning busy icon in the corner of the browser, and a progress bar at the bottom of the browser showing 'in progress'. The browser continues to show these signs that the page is in the process of being downloaded until TCP reports the connection failure to the browser after 21s as shown in Figure 5. Although all objects successfully obtained from the server are obtained over the first connection during the time interval t_0 through t_x, the browser does not indicate the end of the page download until $t_z + 21$, when TCP reports the failure of the second connection to the browser.

For the scenario in Figure 5, our study of web browsers indicates that only a partial page download will occur in practice. The browser will never retrieve the first object which would have been retrieved on the second connection. The browser will retrieve all other objects over the first connection, including those objects which would have been obtained over the second connection had it been established. Therefore, one embedded object is strictly associated with the second failed connection and is not obtained.

If the second connection is eventually established, the embedded object associated with the second connection will then be obtained. For example, suppose that the SYN transmitted at $t_z + 9$ was accepted by the server, the connection was established, and an object was requested and obtained over that connection. The end of the client perceived response time would then be the time that the last byte of the response for that object was received by the client.

A variety of SYN drop combinations could occur, across multiple connections causing various effects on the client perceived response time. If all SYNs on the first connection are dropped, then the client is actually denied access to the server. If both connections are established, each after one or more SYN drops, then the TCP exponential backoff mecha-

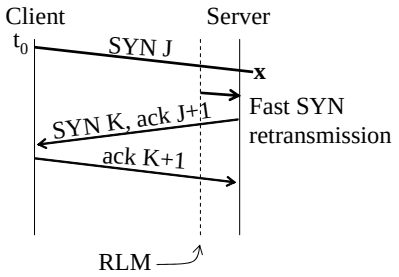

Figure 6: Fast SYN retransmission.

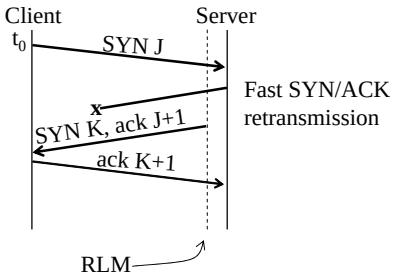

Figure 7: Fast SYN/ACK retransmission.

nism plays an important role in the latency experienced at the remote browser. This effect becomes more pronounced under HTTP 1.0 without KeepAlive where each URL request requires its own TCP connection and the retrieval of each embedded object faces the possibility of SYN drops and possible connection failure. Although the majority of browsers use persistent HTTP, the trend for web servers is to close a connection after a single URL request is serviced if the load is high. Apache Tomcat [34] behaves in this manner when the number of simultaneous connections is greater than 90% of the configured limit, and reduces the idle time if the number of simultaneous connections is greater than 66%. This effectively reduces all transactions to HTTP 1.0 without KeepAlive.

The maximum number of SYN retries that lead to a connection failure defines the connection timeout and depends on the operating system used by the client web browser. In most cases, the default configuration on the number of SYN retries is used. For example, Windows XP operating systems default to two retries, resulting in a connection timeout after 21s. As such, RLM uses 21s in its model as the pageview response time for a web page request that suffers a connection timeout. Other operating systems allow for more SYN retries, so the value we use is conservative as using a larger value would increase the effect of connection failure on response time, exaggerating the benefit of the RLM mechanisms used for managing T_{conn}.

On the other hand, if the browser is painting the screen in a piece-meal manner, indicating that progress is being made, it is more likely that clients will tend to read the pageview as it slowly gets displayed on the screen. This behavior would occur if SYN drops occur on the second connection. In this situation, the pageview response time could exceed 21s.

In all cases, our study of web browsers indicates that packet drops during connection establishment can have a significant, coarse-grained impact on pageview response time. Because of the TCP exponential backoff mechanism, any SYN drop results in a significant increase in T_{conn}. Note that other types of packet drops that occur once a TCP connection is established do not have the same coarse-grain effect. For example, if an HTTP GET request is dropped, the client will retransmit after the retransmission timeout expires, but this timeout value is much smaller than the 3s initial timeout used during connection establishment.

RLM introduces a *fast SYN retransmission* technique that can be used to reduce the coarse-grain effect of SYN drops. Figure 6 depicts the behavior of this mechanism. After a server SYN drop, RLM retransmits the SYN, on behalf of the remote client, at a shorter time interval than the TCP exponential backoff. Since RLM resides within the same complex in which the server exists and is not retransmitting the SYNs over the network, it could at most be considered a locally controlled violation of the TCP protocol. The net effect is that a connection is established as soon as the server is able to accept the request. Since dropping a SYN at the server requires little processing, the overhead of this approach on the server complex is minimal, even when the server is loaded. Nevertheless, the retransmission gap can be adjusted based on the current load or the number of active simultaneous connections.

RLM also introduces a *fast SYN/ACK retransmission* technique that can be used to reduce the coarse-grain effect of SYN/ACK drops. SYN/ACKs dropped in the network cause the same latency effect as a SYN dropped at the server. From the client perspective, there is no difference between a SYN dropped at the server and a SYN/ACK dropped in the network; a SYN/ACK does not arrive at the client and the TCP exponential backoff mechanism applies. Figure 7 depicts the behavior of the RLM fast SYN/ACK retransmission mechanism. If RLM does not capture an ACK from the client within a timeout much smaller than the TCP exponential backoff, RLM retransmits the SYN/ACK to the client on behalf of the server. Fast SYN/ACK retransmission violates the TCP protocol by performing retransmissions using a shorter retransmission timeout period than the exponential backoff. One can make several arguments that this is a minor divergence from the protocol. On the other hand, an Internet web site which uses this technique to improve T_{conn} can rightly be labeled as an unfair participant on the Internet. If deployed, the overhead, either in the network or in the remote client is minimal. Referring to Figure 3, both fast SYN and fast SYN/ACK retransmission can be applied during state transitions 1→2 and 7→8 to reduce the critical path T_{conn}.

5. TRANSFER LATENCY MANAGEMENT

Another key type of latency that RLM must manage is the TCP transfer latency $T_{transfer}$, which can become a dominant component of response time when the network connection between the client and the server is the bottleneck. $T_{transfer}$ is known to be a function of object size, network round trip time (RTT) and packet loss rate:

$$T_{transfer} = f(size, RTT, loss) \quad (1)$$

Several analytic models of $f(size, RTT, loss)$ have been developed [28, 10, 32]. Figure 8 depicts the transfer latency function defined by Cardwell [10], for realistic Internet conditions of an RTT of 80ms and loss rate of 2% [39]. The line indicates the expected time (y-axis) it will take to transfer an object of the given size (x-axis). For smaller objects, in this case less than 10 packets in size, the transfer latency is dominated by TCP slow start behavior, the logarithmic part

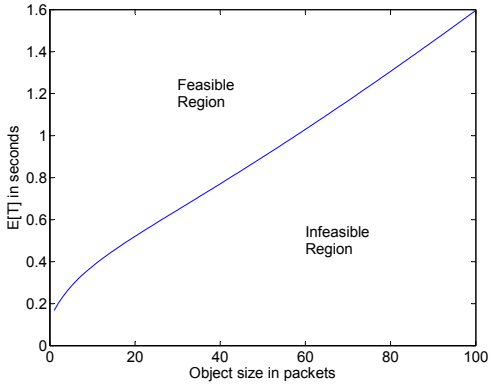

Figure 8: Cardwell et al. Transfer Latency Function f for 80ms RTT and 2% loss rate.

of the graph. For larger objects, the transfer latency is dominated by TCP steady-state behavior, the near-linear part of the graph. Cardwell's function is a model of the expected amount of time required, not the minimum time. The farther a point is from the line, the less likely it is to occur in practice. For example, it is extremely unlikely that an object of size 50 packets can be transferred in under 1 second if the RTT is 80ms and the loss rate is 2%. We labeled the region below the line as *infeasible*. The model predicts that under higher loss rates and longer RTT, reducing object size can reduce $T_{transfer}$ by half.

Since RTT and loss rate are a function of the end-to-end path from client to server through the Internet and therefore uncontrollable, RLM is left with varying the response size as a control mechanism for affecting $T_{transfer}$. RLM accomplishes this using two simple techniques:

1. *Embedded object rewrite*: Translate a request for a large image into a request for a smaller image. Capture the HTTP request packet, if the request is for a large image then modify the request packet by overwriting the URL so that it specifies a smaller image, and then pass the request onto the server.

2. *Embedded object removal*: Remove references to embedded objects from container pages. Capture the HTTP response packets, if the response is for a container page then modify the response packet by overwriting references to embedded objects with blanks, and then pass the request packet onto the client.

Embedded object rewrite retrieves an embedded object but one of much smaller size than the original, reducing the response size and $T_{transfer}$ for that object. The trade-off is that the quality of the content is affected since the client will see a lower quality image. By modifying the client to server HTTP request, RLM can decide on a per request basis, in the middle of a pageview download, whether or not to change the requested object size. This presumes the existence of smaller objects; for some web sites, maintaining all or some of their images in two or more sizes may not be possible. This technique can also be applied to dynamic content, where a less computationally expensive CGI is executed in place of the original, or the arguments to the CGI are modified (e.g. a search request has its arguments changed to return at most 25 items instead of 200).

Embedded object removal entirely avoids retrieving an embedded object, eliminating $T_{transfer}$ for that embedded object. This has a greater latency reduction effect than the embedded object rewrite, but may further reduce the quality of the web content displayed. Instead of viewing thumbnail images, the client only sees text. Unlike embedded object rewrite which can be applied for any image retrieval during pageview download, the decision on whether or not to blank out embedded objects in the container page can only be made at one point in the pageview download, when the container page is being sent from the server to the client, which is transition 3→4 in Figure 3.

These content adaptation techniques may also be able to reduce other types of latencies. Embedded object rewrite can reduce T_{server} on the server and T_{render} at the client when the smaller object is also faster to serve and render. Embedded object removal can eliminate T_{conn} for an embedded object if a connection is no longer required, and can eliminate T_{server} and T_{render} if the object no longer needs to be served or displayed. However, the resulting change in the pageview response time due to these latencies will depend on whether the server delivering embedded objects or the client rendering them are the bottleneck, the latter being unlikely with modern PC clients.

RLM adapts content by simply modifying a packet and forwarding the modified version; it does not need to keep buffers of packet content. RLM is not a proxy, and as such, must ensure the consistency of the sequence space for each connection. This means that changing the HTTP request/response is constrained by the size and amount of white space in each packet, and the checksum value must be recomputed as it changes with the change in payload.

6. EXPERIMENTAL RESULTS

We implemented RLM as a set of kernel modules that can be loaded into an inexpensive, off-the-shelf PC running Linux. Our kernel module approach is based on our previous work which demonstrated significant performance scalability benefits for executing within kernel space [27]. We present some results using RLM to manage client perceived response times in both single and multiple service class environments using TPC-W [33], a transactional web e-Commerce benchmark which emulates an online book store, running on a three-tier web architecture.

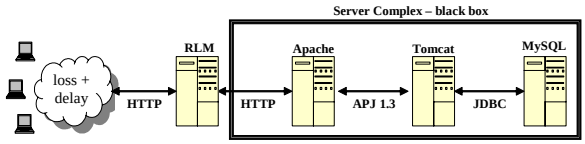

Figure 9: Experimental testbed.

Figure 9 shows our experimental testbed. It consists of seven machines: three web clients, one RLM appliance, and three servers functioning as a three-tier web architecture. Apache 2.0.55 was installed as the first tier HTTP server and was configured to run up to 1200 server threads using the worker multi-processing module configuration. Apache Tomcat 5.5.12 [34] was employed as the second tier application server (servlet engine) and was configured to maintain a pool of 1500 to 2000 AJP 1.3 server threads to service requests from the HTTP server, and a pool of 1000 persis-

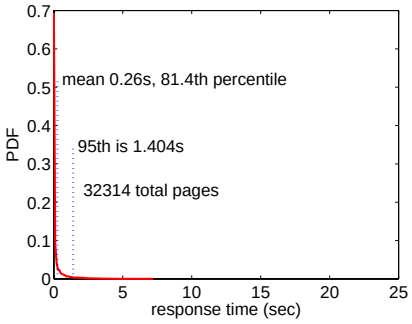

Figure 10: 0.3ms RTT, 0% loss.

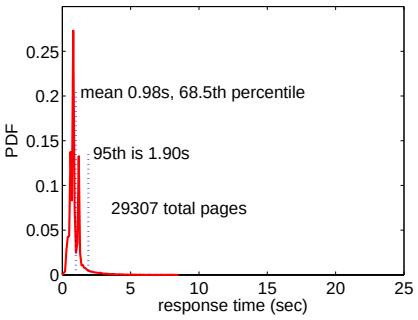

Figure 11: 80ms RTT, 0% loss.

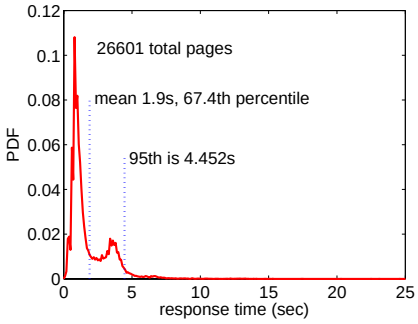

Figure 12: 80ms RTT, 4% loss.

tent JDBC connections to the database server. MySQL 1.3 was employed as the third tier database (DB) server and was set to the default configuration with the exception that the max_connections was changed to accommodate the 1000 persistent connections from Tomcat.

Each of the three client machines was an IBM IntelliStation M Pro 6868 with a 1GHz Pentium 3 CPU and 512MB RAM. The RLM machine was an IBM IntelliStation 6850 with a 1.7GHz Xeon CPU and 768MB RAM. The Apache machine was an IBM IntelliStation M Pro 6868 with a 1 GHz Pentium 3 CPU and 1GB RAM. The Tomcat machine was an IBM IntelliStation M Pro 6849 with a 1.7GHz Pentium 4 CPU and 768MB RAM. The MySQL machine was an IBM IntelliStation 6850 with a 1.7GHz Xeon CPU and 768MB RAM. All machines were running RedHat Linux, with the DB server running a 2.6.8.1 Linux kernel and the other machines running a 2.4.20 Linux kernel. The machines were connected via 100Mbps Fast Ethernet Netgear, CentreCOM, and Dell switches. We installed a modified version of the rshaper [30] bandwidth shaping tool on each of the three client machines to emulate wide-area network conditions in terms of transmission latencies and packet loss.

We used a popular Java implementation of TPC-W [35] for our workload, but made two important modifications to the client emulated browser (EB) code to make it behave like a real web browser such as Microsoft Internet Explorer. First, we modified the EB code to use two persistent parallel connections as shown in Figure 2 over which the container object and embedded objects are retrieved. These connections were not closed by the client but remained open during the client think periods (unless closed by the server). The original EB sent HTTP/1.1 request headers but actually used one connection for each GET request, effectively emulating HTTP/1.0 behavior by opening a connection, sending the request, reading the response and closing the connection. Second, we modified the EB to behave under connection failure as shown in Figure 5. We also used IP aliasing so that each individual EB could obtain its own unique IP address.

The TPC-W e-Commerce application consists of a set of 14 servlets. Each pageview download consists of the container page and a set of embedded GIF images. All container pages are built dynamically by one of the 14 servlets running within Tomcat. First, the servlet performs a DB query to obtain a list of items from one or more DB tables, then the container page is dynamically built to contain that list of items as references to embedded images. After the container page is sent to the client, the client parses it to obtain the list of embedded images, which are then retrieved from Apache. As such, all images are served by the front end Apache server, and all container pages are served by Tomcat and MySQL.

6.1 Response Time Distribution

We first present measurements running TPC-W under ideal conditions of light load and no network loss or delay; we then add network loss and delay to see the effect this has on the response time distribution. We use 200 clients which keeps the DB server, which is the bottleneck resource in our multi-tier complex, at only 60% utilization.

Figure 10 shows the response time distribution under ideal network conditions of minimal delay and zero loss along with the mean and 95th percentile client perceived response time and the total number web pages served. This scenario is often used for web server performance benchmarking and QoS experimentation, but is very unrealistic for an Internet web site. Figure 11 shows the response time distribution for the same experiment under 80ms RTT and zero packet loss. The additional RTT shifts and spreads the distribution to the right as the transfer latency becomes more significant for larger pageviews.

Figure 12 shows the response time distribution under more realistic network conditions of 80ms RTT and 2% network loss in each direction; studies have shown that the packet loss rate within the Internet is roughly 1-3% [39]. The distribution shifts further to the right and a clearly distinguishable spike occurs just after 3s. This is attributed to either the first or second connection of the pageview having an initial SYN or SYN/ACK drop in the network. The response time distribution in Figure 12, not the one shown in Figure 10, is most likely to be the shape of the response time distribution for web clients. Any approach which claims to manage client perceived response time for Internet web service ought to be verified under conditions found in the Internet: network latency and loss. Unless otherwise indicated, all of our experiments are done using the same 80ms RTT network latency and 2% network loss.

6.2 Managing Connection Latency

Table 1 shows how RLM can improve response times by applying fast SYN/ACK retransmission to the same experiment shown in Figure 12 with different SYN/ACK retransmission gaps. For each retransmission gap, we report mean client pageview (PV) response time (RT), mean PV RT speedup versus Figure 12, 95th percentile PV RT, percentage of pages downloaded with greater than 3s response time, and total number of pageviews. For example, the following

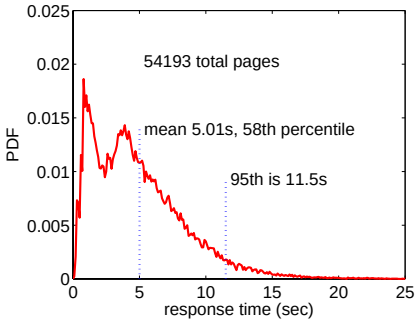

Figure 13: Unmanaged heavy load.

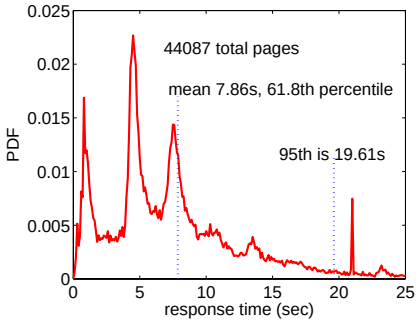

Figure 14: MaxClients load shedding.

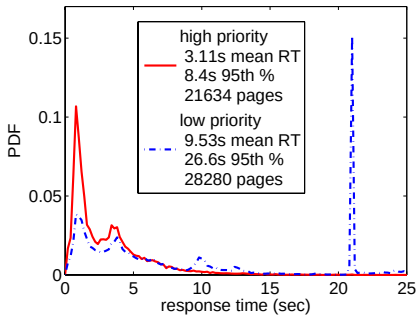

Figure 15: Low priority penalties.

SYN/ACK gap	mean PV RT	mean speedup	95^{th}% PV RT	> 3s PV RT	total pages
3s	1.9s	0%	4.45s	22.36%	26601
1s	1.72s	9.5%	4.22s	18.17%	27001
500ms	1.64s	13.7%	4.09s	16.05%	27287
10ms	1.58s	16.8%	4s	15.42%	27455

Table 1: Fast SYN/ACK retransmission.

RLM rule is used for a 500ms retransmission gap:

IF IP.SRC == *.*.*.* THEN
FAST SYN/ACK GAP 500ms

Fast SYN/ACK retransmission results in a modest reduction in the mean response time, as much as 16.8% for the smallest retransmission gap of 10ms. More importantly, the technique results in a much larger reduction in the number of pages that have higher response times, reducing the number of pages with more than 3s response time by over 30%. In general, we would not expect this technique to significantly affect the mean response time, but rather significantly affect those pageviews that experience a network SYN/ACK drop.

6.3 Managing Load and Admission Control

Figure 13 shows the response time distribution for running TPC-W with 550 clients, causing heavy load. The mean client perceived response time increased to 5s from the 1.9s for 200 clients shown in Figure 12. No SYN drops are occurring at the server complex. The only SYNs being dropped are those being lost in the network, so the percentage of SYN drops is the same for both the light and heavy load experiments shown in Figures 12 and 13, respectively. Bandwidth is at low utilization throughout the entire testbed. The increase in response time is due to increased CPU utilization within the multi-tier complex.

In such a scenario, it is usually desirable to apply a load shedding technique to prevent the web server from overloading or to simply improve response time by reducing the load. A simple and common load shedding mechanism is to manipulate the Apache `MaxClients` setting [15, 23, 37]. `MaxClients` is an upper bound on the number of httpd threads available to service incoming connections; it limits the number of simultaneous connections being serviced by Apache.

Figure 14 shows the result of setting `MaxClients` to 400 for the same workload shown in Figure 13. The spike at 5s in the distribution represents those pageviews which incurred an initial SYN drop resulting in a 3s timeout on one of the two EB connections to the server (in addition to the 2s baseline latency shown in Figure 12). The spike at 8s, which is barely visible in Figure 12 but pronounced in Figure 14, represents those pageviews which incurred a 3s timeout on both connections to the server. The spike at 21s represents those clients which experienced a connection failure. Table 2 depicts the results for various limits on the number of simultaneous connections served.

Max Clients	mean PV RT	95^{th}% PV RT	Tomcat RT	total pages	server SYN drops
600	5.01s	11.6s	3.14	54193	0%
500	5.28s	11.9s	1.013	53038	4.8%
400	7.86s	19.6s	0.405s	44087	18.2%
300	12.3s	28.5s	0.155s	34440	30.7%
200	19.1s	40.5s	0.068s	25894	43.5%

Table 2: Load shedding via connection throttling.

We instrumented the TPC-W servlets to capture their response time by taking a timestamp when the servlet was called and a timestamp when the servlet returned; this covers the time it takes to build the container page, including the DB query but does not include the time to connect to the server complex or transmit the response. Table 2 shows under Tomcat response time that as the number of simultaneous connections decreases, the mean time to query the DB and create the container page decreases. However, when measured in terms of pageviews perceived by the client and including those pages which experienced the default admission control drops, the overall mean pageview response time actually increases. Some clients are experiencing response times which can be considered as better than required while other clients are experiencing significant latencies due to SYN drops. The results demonstrate that this common form of load shedding is ineffective at reducing client perceived response times. Furthermore, the significant effect that SYN drops have on the response time distribution makes providing service level agreements based on meeting a threshold for the 95^{th} percentile impossible to achieve.

A common alternative to changing the Apache MaxClients is to perform SYN throttling to control the offered load on the system. We apply this technique in the context of a multi-class QoS environment. Many web sites demand the ability to support different users with multiple classes of service and response time, such as providing buyers with better performance than browsers at an e-commerce site. It is often desirable to maintain a specific response time threshold for a certain class of clients. Given a finite set of resources under heavy load, high priority clients are expected to receive

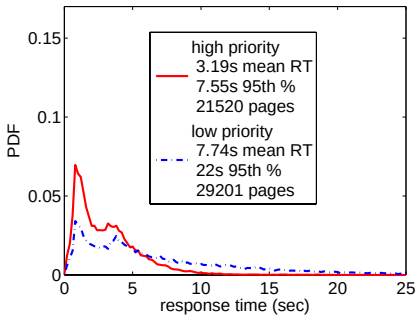

Figure 16: Fast SYN improvement.

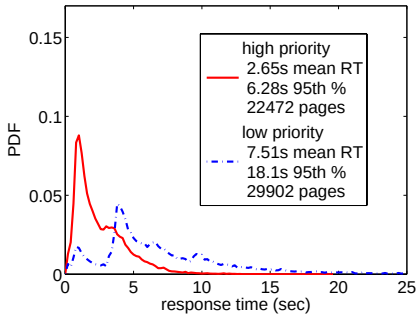

Figure 17: Widening think time gap.

better response time than if all clients were treated equally. Similarly, low priority clients will suffer worse response time than if all cilents were treated equally.

We ran TPC-W with 550 clients as we did in Figure 13, but divided the clients into one-third high priority clients from subnet 10.4.*.* and two-thirds low priority clients from other subnets. We perform typical SYN throttling (i.e. admission control) by dropping SYNs arriving from low priority clients when the high priority clients are exceeding their response time threshold. We employ RLM using the following rule:

IF IP.SRC ! = 10.4.*.* AND RT_HIGH > 3.0s THEN
DROP SYN

Figure 15 shows that mean response time for the 184 high priority clients was roughly 3s, but at a heavy cost to the 366 low priority clients. The vertical spike at 21s for the low priority clients indicates the set of connection failures experienced by those clients. From Figure 12 we see that 200 clients alone receive 2s mean response time. As such, our 184 high priority clients have processing to spare for the low priority clients. But the high retransmission penalty significantly affects the response time of the low priority clients. RLM can improve this situation by using fast SYN and SYN/ACK retransmission. Figure 16 shows the effect of the following rule:

IF IP.SRC == *.*.*.* THEN
FAST SYN + SYN/ACK GAP 500ms
IF IP.SRC ! = 10.4.*.* AND RT_HIGH > 3.0s THEN
DROP SYN
HALT FAST SYN

In enforcing this rule, if the mean response time for the high priority clients exceeds 3s, then incoming SYNs from low priority clients are dropped by RLM and existing low priority fast SYN retransmissions are temporarily halted. The moment RT_HIGH $<$ 3.0s, the low priority fast SYN retransmission is resumed and new SYNs from low priority clients are passed to the server. Low priority clients have their requests processed without them waiting the full TCP retransmission timeout periods. This lead to a 23% improvement for low priority clients, while maintaining essentially the same response time and throughput of the high priority clients. Most importantly, the spike at 21s in Figure 15 is gone with the removal of the large number of connection failures experienced by the low priority clients.

Figure 17 depicts an alternative distribution to Figure 16 based on using the same rule with the addition that all *initial* SYNs on the *first* connection of the pageview from low priority clients are dropped, with a fast SYN retransmitted 3s later and 500ms subsequently there after. By dropping *all* initial SYNs for low priority clients we effectively increased their think time, the interarrival time between container page requests. This shifts the distribution to the right and has the effect of improving the 95th% of both service classes. Note that some low priority pages are served very fast. These pages are re-using the TCP connection for the container page, and hence unaffected by SYN manipulation. Reducing the arrival rate of low priority clients resulted in a reduction in load on the MySQL server, which in turn reduced server time and client response time.

Application of this technique warrants further investigation. Over-use could lead to livelock [24] or create more work for the system, offsetting the benefits of reducing the drop latencies (although, in an e-commerce environment the front-end web server is not the bottleneck). We have not observed any negative effects on the TCP endpoints (server/client) with respect to their ability to measure RTT and enforce their TCP timeouts/retransmissions.

6.4 Managing Transfer Latency

We now consider managing transfer latency under situations of variable and larger RTT. We run TPC-W with 200 clients so that the multi-tier complex is under modest load and modified our environment by splitting our clients into three groups, one having 160ms RTT, another with 220ms RTT and the third with 300ms RTT. The resulting mean client perceived response times were roughly 3s, 4s, and 5s, respectively. In this environment, nothing in the server complex is overloaded, and no server-side SYN drops are occurring. As such, load shedding performed at the server will not improve response times.

Figure 18 shows the impact of embedded object removal on mean response time as a function of the number of embedded objects removed and retrieved. In this graph, the x-axis represents the number of embedded objects retrieved and therefore not removed. For example, when x is 3, the first 3 embedded objects are retrieved while the rest of the objects are removed from the container page. No page had more than 11 objects per page, so the rightmost points in Figure 18 correspond to full downloads. The leftmost points correspond to all embedded objects being removed, reducing mean response time to near 1s, using the following RLM rule:

IF IP.SRC = *.*.*.* THEN REMOVE EMBEDS

Without embedded object removal, the difference in RTT

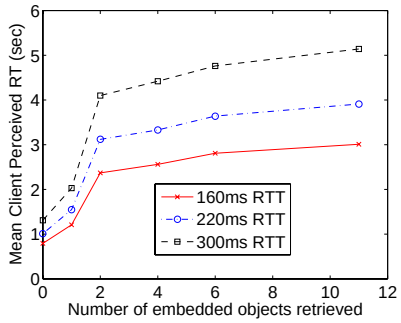

Figure 18: Embedded object removal.

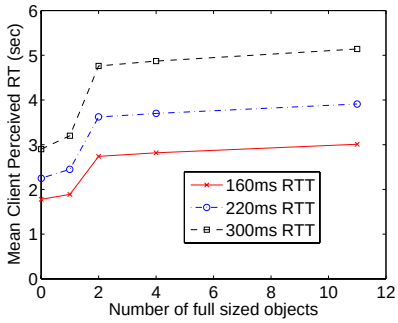

Figure 19: Embedded object rewrite.

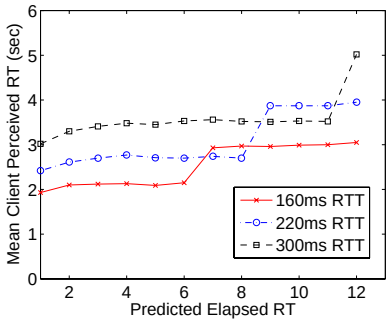

Figure 20: Predicted elapsed time.

separates out the clients into three service classes when only one service class is desired. Figure 18 shows that different numbers of embedded objects can be removed for clients with different RTTs to provide similar response times for all clients. For example, clients with different RTT can be given a mean response time of 3s by doing full downloads for the 160 ms RTT clients, only allowing 220ms RTT clients to download two embedded objects, and having half the 300ms RTT clients download one object and receive 2s response time while the other half download two objects and receive 4s response time. The mix of different numbers of embedded objects for the 300ms RTT clients is due to the discrete nature of the technique: either an object is obtained or it is not obtained. Although discrete, embedded object removal has the advantage over rewrite in that multiple copies of the same object need not be maintained. This is an issue if disk space is limited or charged by use.

Note the large jump in Figure 18 when the second embedded object is downloaded. While this is partly due to the overhead in opening the second connection, the key reason is that for roughly 18% of the pageviews in TPC-W, the second embedded object is a large 256KB GIF image. Since it is being downloaded as the first object on the second connection, not only does it incur T_{conn} but also TCP slow-start. By configuring RLM to remove only the second image, the response time dropped to 2.19s for the 160ms RTT clients, 2.85s for the 220ms RTT clients, and 3.68s for the 300ms RTT clients. The curves in Figure 18 are relatively flat after the second object. As more objects are downloaded on the same persistent connection, the TCP window size increases. In addition, fewer pages have larger number of embedded objects. Roughly 75% of the pages contain 9 objects, and 18% of the TPC-W pages contain 10 or more. If the typical e-commerce web site has a similar shape, then removal/rewrite could be applied in this manner, bottom to top, for the pages with the most items; this implies fewer clients will be affected.

Figure 19 shows the corresponding results for embedded object rewrite. Embedded object removal is more effective at reducing response time than embedded object rewrite, but the effect is coarse-grained. The removal of the references to embedded objects must occur during the transition 3→4 in Figure 3, essentially eliminating states 6 through 18. In contrast, embedded object rewrite can be applied at a finer-granularity as different parts of the page are downloaded. To reduce response time, one would like to apply a rule such as:

IF RT > 2s THEN REWRITE EMBEDS

However, this may not be effective. Referring back to Figure 3, it is at node 5 that the browser obtains the list of embedded objects to obtain. In our current scenario, this is after the server, which is under light load, returns the container page. At this point, the elapsed response time is relatively short, less than 1s. It is at this moment that the EB opens the second connection and may request the large image which greatly increases the response time. Even if we decide after 1s to rewrite the remaining objects, the time required to finish downloading the large image will extend the pageview response time to beyond 1s. Indeed, the result we obtain is a response time of 2.8s, 3.52s and 4.55s for the 160ms, 220ms and 300ms RTT clients, respectively; this matches the download of two full size images in Figure 19.

This indicates the need to predict the latency contribution that a request will have to the pageview RT. Figure 20 shows the results of rewriting embedded objects if the predicted $T_{transfer}$ for that object would cause the pageview to exceed the specified elapsed time. RLM keeps an average of the $T_{transfer}$ for image downloads under different size, RTT and loss groupings, which is comparable to Equation 1. Note this does not include T_{render}, and as such is an underestimate of the latency associated with the image. For an RTT of 160ms, 220ms and 300ms the large image is predicted to have a $T_{transfer}$ value of 6.17s, 8.11s and 11.05s, respectively. Notice that at a 1s threshold Figure 20 matches the response time as seen in Figure 19 when rewriting all images - hence correctly removing the large image for the short threshold. As the threshold increases, the predictor allows the large image to be included in the pageview when it no longer affects the RT. This is seen at 7s, 9s and 12s for 160ms, 220ms and 300ms clients. This technique tends to shorten the tail on the response time distribution by removing embedded objects which take longer to download.

Figure 21 shows the effect of embedded object removal as compared to full page downloads as the number of clients increase. Once the number of clients reaches 550 the time to download full pages vs. pages with no embedded objects is the same. At this point, when full pages are being downloaded the 550 clients are spending half their time at the DB server obtaining the container page and half their time at Apache retrieving the embedded objects - this means the DB server is serving roughly 275 requests simultaneously. When the embedded images are removed from the container page,

the client spends no time at Apache and all the time at the DB server - effectively doubling the load on the DB server so that the DB server is serving roughly 550 requests simultaneously. The extra load on the DB server causes longer delay in serving the container page. As a reference point, the dotted line in Figure 21 shows the split between the time spent on the container page (below the line) and the embedded objects (above the line). The time required to obtain the embedded objects from Apache increases the time between DB requests to the MySQL server. As such, the complex could return a full pageview or an empty pageview at the same RT. Note that the think time in both cases is the same. This clearly shows that attempting to managing one portion of the response time instead of the overall pageview response time may not be effective.

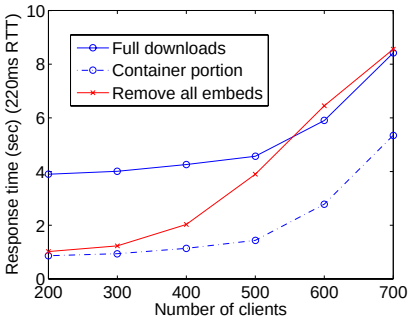

Figure 21: Full vs. Empty, RT.

7. RELATED WORK

Several other systems have been developed to manage response times. eQoS [37] seeks to manage pageview response time by adjusting the number of simultaneous connections being serviced (similar to Hellerstein et al. [15, 23]), but tracks the activity between client and Apache in user space within the web server by intercepting socket level transactions. As such, it is unable to detect latencies due to SYN drops, time spent in kernel queues, and network RTT and loss. It simply manages the server response time, not the remote client perceived response time. Quorum [9] also takes a blackbox approach towards the server complex, but is a proxy front-end that throttles URL requests into the server. Quorum is driven by the per URL server response time and reports results for only successful requests, ignoring the dropped URL requests and their impact on the pageview response time. RLM is not a user space proxy but a packet level system, so it can track latencies not only at the HTTP level but within TCP. Several other approaches manage service quality on a per URL basis [29, 13, 8, 3, 1, 36].

Other approaches have considered measuring performance in terms of sessions, not pageviews. Cherkasova et al. [12] performed admission control with the goal of maximizing the number of successful sessions and limiting the amount of resources wasted on rejected sessions. Kihl and Widell [22] showed that this reduces the number of angry customers (Bhatti et al. has studied user's tolerance for delay in [7]). We see our technique complementing this work; RLM could be extended to manage individual pageview downloads in the context of a user session.

Other mechanisms for controlling parts of web response time have been proposed. Elnikety et al. [16] developed a DB proxy that manages load on the DB server by performing admission control on DB queries being sent from the second tier application server to the third tier DB server, but do not report the effect this has on the content of the pageview seen at the web browser. Crovella et al. [14] manage transfer latency by scheduling outbound bandwidth based on shortest remaining response length, where the byte count of the response is used as the response length; this is applied when the bandwidth at the server is the bottleneck and does not take into account RTT and loss. Jamjoom and Shin [20] showed that to minimize the per connection latency associated with SYN drops, the best approach in steady-state is to accept the initial SYN or drop it and all its retransmissions. With Pillai [19] they presented *Abacus*, a modified token bucket filter, and showed under simulation how it smoothed out the synchronization effects associated with SYN retransmissions. In contrast, RLM demonstrates the benefits of SYN retransmissions for obtaining a specified response time in the context of a dynamic, realistic e-commerce workload.

Similar techniques to those used in RLM have been used in other contexts. In particular, Abdelzaher and Bhatti [2] created an agent to perform content adaptation from user space on the web server machine by swapping the content directory that the web server reads. This system was URL based and driven by CPU or bandwidth utilization. Other applications of content adaptation include multimedia transcoding [11] and content negotiation [31]. A fast retransmission mechanism violating TCP was presented in the context of wireless networking [4, 5] to alleviate latencies associated with loss due to the physical medium. However, this was not applied to SYN and SYN/ACK processing but only to data transfer over established connections.

8. CONCLUSIONS AND FUTURE WORK

Remote Latency-based Management (RLM) is a new approach for managing the client perceived response time of a web services infrastructure using only server-side techniques. RLM tracks each pageview download as it occurs and makes control decisions at each key juncture in the context of a specific pageview download. RLM introduces fast SYN and SYN/ACK retransmission mechanisms and uses content adaptation in the form of embedded object removal and rewrite. These techniques can be applied based on control decisions during the pageview download. We have implemented RLM as a stand-alone Linux PC appliance that simply sits in front of a web server complex and manages response time, without any changes to existing web clients, servers, or applications. Our results demonstrate the importance of measuring client perceived pageview response time in managing web server performance, the limitations of existing admission control techniques, and the benefits of RLM's mechanisms for controlling response time to manage an overloaded web server complex and to mitigate the negative impact of network latencies and loss.

RLM provides a starting point for future work in developing a comprehensive management scheme that can provide a range of policies for controlling client perceived response times, and can meet different service level objectives for different classes of service. Many other packet manipulation techniques can be explored in this context, including manipulating drop, delay or retransmission of URL requests and responses. While RLM provides a blackbox approach

to management when modifications to a web server complex are not possible, its core management framework can be used with other more invasive mechanisms that can be deployed in the web server complex to achieve a greater degree of control of resources to manage overall response time.

9. ACKNOWLEDGMENTS

This work was supported in part by NSF grants ANI-0117738 and ANI-0240525, and IBM SUR and Faculty Awards. Thanks to Li Zhang, Erich Nahum, Hani Jamjoom and John Tracey for discussion, insight and support.

10. REFERENCES

[1] T. Abdelzaher, K. G. Shin, and N. Bhatti. Performance Guarantees for Web Server End-systems: A Control Theoretical Approach. *IEEE Transactions on Parallel and Distributed Systems*, 13(1), January 2002.

[2] T. F. Abdelzaher and N. Bhatti. Web Content Adaptation to Improve Server Overload Behavior. *Computer Networks*, 31(11-16):1563–1577, 1999.

[3] J. Almeida et al. Providing Differentiated Levels of Service in Web Content Hosting. In *Workshop on Internet Server Performance Conference*, June 1998.

[4] H. Balakrishnan et al. A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. *IEEE/ACM Transactions on Networking (TON)*, 5(6):756–769, December 1997.

[5] H. Balakrishnan, S. Seshan, and R. Katz. Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks. *ACM Wireless Networks*, 1(4):469–481, December 1995.

[6] P. Barford and M. Crovella. Critical Path Analysis for TCP Transactions. In *ACM SIGCOMM 2000*, p. 127–138, Stockholm, Sweden, August-September 2000.

[7] N. Bhatti, A. Bouch, and A. Kuchinsky. Integrating User-Perceived Quality into Web Server Design. In *9th International World Wide Web Conference*, Amsterdam, Netherlands, May 2000.

[8] N. Bhatti and R. Friedrich. Web Server Support for Tiered Services. *IEEE Network*, 13(5):6764–6771, September-October 1999.

[9] J. M. Blanquer et al. Quorum: Flexible Quality of Service for Internet Services. In *NSDI 2005*, p. 159–174, Boston, MA, May 2005.

[10] N. Cardwell, S. Savage, and T. Anderson. Modeling TCP Latency. In *IEEE Infocom*, volume 3, p. 1742–1751, 2000.

[11] S. Chandra, C. Ellis, and A. Vahdat. Differentiated Multimedia Web Services Using Quality Aware Transcoding. In *IEEE Infocom*, Tel-Aviv, Israel, March 2000.

[12] L. Cherkasova and P. Phaal. Session Based Admission Control: a Mechanism for Improving Performance of Commercial Web Sites. In *IWQoS 1999*, p. 226–235, London, UK, May 1999.

[13] I. Cohen et al. Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control. In *OSDI 2004*, p. 231–244, San Francisco, CA, December 2004.

[14] M. Crovella, R. Frangioso, and M. Harchol-Balter. Connection Scheduling in Web Servers. In *USITS 1999*, p. 243–254, Boulder, CO, October 1999.

[15] Y. Diao, J. L. Hellerstein, and S. Parekh. Optimizing Quality of Service using Fuzzy Control. In *Distributed Systems Operations and Management*, 2002.

[16] S. Elnikety et al. A Method for Transparent Admission Control and Request Scheduling in E-Commerce Web Sites. In *WWW2004*, p. 276–286, New York, NY, May 2004.

[17] Y. Fu, L. Cherkasova, W. Tang, and A. Vahdat. EtE: Passive End-to-End Internet Service Performance Monitoring. In *USENIX Conference Proceedings*, 2002.

[18] E. Hu et al. Adaptive Fast Path Architecture. *IBM Journal of Research and Development*, 45(2):191–206, April 2001.

[19] H. Jamjoom, P. Pillai, and K. G. Shin. Re-synchronization and Controllability of Bursty Service Requests. *IEEE/ACM Transactions on Networking*, 12(4):582–594, August 2004.

[20] H. Jamjoom and K. G. Shin. Persistent Dropping: an Efficient Control of Traffic Aggregates. In *ACM SIGCOMM 2003*, p. 287–298, Karlsruhe, Germany, August 2003.

[21] M. F. Kaashoek at al. Application Performance and Flexibility on Exokernel Systems. In *OSDI*, Saint-Malo, France, October 1997.

[22] M. Kihl and N. Widell. Admission Control Schemes Guaranteeing Customer QoS in Commercial Web Sites. In *IFIP/IEEE NETCON*, Paris, France, October 2002.

[23] X. Liu et al. Online Response Time Optimization of Apache Web Server. In *IWQoS*, p. 461–478, 2003.

[24] J. Mogul and K. K. Ramakrishnan. Eliminating Receive Livelock in an Interrupt-Driven Kernel. *ACM TOCS*, 15(3):217–252, 1997.

[25] E. Nahum, M. Rosu, S. Seshan, and J. Almeida. The Effects of Wide-Area Conditions on WWW Server Performance. In *ACM SIGMETRICS*, 2001.

[26] D. Olshefski, J. Nieh, and D. Agrawal. Using Certes to Infer Client Response Time at the Web Server. *ACM TOCS*, 22(1):49–93, February 2004.

[27] D. Olshefski, J. Nieh, and E. Nahum. ksniffer: Determining the Remote Client Perceived Response Time from Live Packet Streams. In *OSDI 2004*, p. 333–346, San Francisco, CA, December 2004. USENIX.

[28] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A Simple Model and Its Empirical Validation. *ACM SIGCOMM Computer Communication Review*, 28(4):303–314, 1998.

[29] P. Pradhan et al. An Observation-based Approach Towards Self-managing Web Servers. In *IWQoS 2002*, p. 13–22, Miami, FL, May 2002. IEEE/ACM.

[30] A. Rubini. rshaper. HTTP://www.linux.it/ rubini/software/index.html.

[31] S. Seshan, M. Stemm, and R. Katz. Benefits of Transparent Content Negotiation in HTTP. In *IEEE GLOBECOM 98 Internet Mini-Conference*, Sydney, Australia, Nov. 1998.

[32] B. Sikdar, S. Kalyanaraman, and K. Vastola. Analytic Models and Comparative Study of the Latency and Steady-state Throughput of TCP Tahoe, Reno and SACK. In *IEEE GLOBECOM*, p. 100–110, San Antonio, TX, November 2001.

[33] The Transaction Processing Council (TPC). http://www.tpc.org/tpcw.

[34] Tomcat 5.5. http://www.jakarta.apache.org/tomcat.

[35] TPC-W Java Implementation. http://mitglied.lycos.de/jankiefer/tpcw.

[36] T. Voigt et al. Kernel Mechanisms for Service Differentiation in Overloaded Web Servers. In *USENIX 2002*, p. 189–202, 2001.

[37] J. Wei and C.-Z. Xu. *e*QoS: Provisioning of Client-Perceived End-to-End QoS Guarantees in Web Servers. In *IQWoS 2005*, Passau, Germany, June 2005.

[38] C. Williamson and Q. Wu. A Case for Context-Aware TCP/IP. *ACM SIGMETRICS Performance Evaluation Review*, 29(4):11–23, 2002.

[39] Y. Zhang et al. On the Constancy of Internet Path Properties. *ACM SIGCOMM Internet Measurement Workshop (IMW)*, San Francisco, CA, November 2001.