Ensuring Latency Targets In Multiclass Web Servers

Vikram Kanodia and Edward W. Knightly, *Member*, *IEEE*

Abstract—Two recent advances have resulted in significant improvements in web server quality-of-service. First, both centralized and distributed web servers can provide isolation among service classes by fairly distributing system resources. Second, session admission control can protect classes from performance degradation due to overload. The goal of this work is to design a general "front-end" algorithm that uses these two building blocks to support a new web service model, namely, multiclass services which control response latencies to within prespecified targets. Our key technique is to devise a general service abstraction to adaptively control not only the latency of a particular class, but also to bound the interclass relationships. In this way, we capture the extent to which classes are isolated or share system resources (as determined by the server architecture and system internals) and hence their effects on each other's QoS. For example, if the server provides class isolation (i.e., a minimum fraction of system resources independent of other classes), yet also allows a class to utilize unused resources from other classes, the algorithm infers and exploits this behavior, without an explicit low level model of the server. Thus, as new functionalities are incorporated into web servers, the approach naturally exploits their properties to efficiently satisfy the classes' performance targets. We validate the scheme with trace driven simulations.

Index Terms—Web servers, QoS, admission control, multiclass, statistical envelopes.

1 Introduction

The explosive growth in web traffic [17] has led to excessive latencies due to overloaded web servers. Consequently, reducing and controlling server latencies is a key challenge for delivering end-to-end quality-of-service.

Towards this end, two key mechanisms have been introduced to improve web QoS. First, *admission control* has been proposed as a mechanism to prevent web servers from entering overload situations [4], [10], [16]. Specifically, by admitting new sessions only if the measured load is below a prespecified threshold, admission control can prevent the server from entering a regime in which latencies are excessive, or session throughput collapses due to dropped requests and aborted sessions.

Second, web servers can now provide *performance isolation and differentiation* among the different service classes hosted by the site. In particular, a server may support a number of service classes which may represent different classes of users or different applications (news, email, static documents, dynamic content, etc.). Whether such classes are supported in a single-node server or a distributed cluster, mechanisms devised in [3], [6], [7] and [2], respectively, can ensure that each service class receives a certain share of system resources (disk, CPU, memory, etc.). Moreover, by appropriately weighting the share of system resources, *differentiation* among service classes is achieved. Similarly, as delays are also incurred in the system's request queues, prioritization of incoming requests can further differentiate the performance among classes [4], [12].

Thus, differentiation and isolation can be achieved by prioritized scheduling of system resources, and protection from overload can be achieved by admission control. However, even if taken together, these two mechanisms cannot ensure that a request's targeted delay will be satisfied. Consequently, because end-to-end latency is a key component of user-perceived quality-of-service, new mechanisms are needed to ensure that the service class' request delays are limited to within the targeted value.

In this paper, we introduce a new framework for multiclass web server control which can satisfy per class latency constraints and devise an algorithm termed Latency-Targeted Multiclass Admission Control (LMAC). Our key technique is to design a scheme within a general framework of request and service envelopes. Such envelopes statistically describe the server's request load and service capacity as a function of interval length, resulting in a high-level service abstraction which circumvents the need to model or measure the components of a request's delay. For example, a request incurs delays in the request queue, CPU processing, memory, disk in the case of cache misses, and so on; individually controlling the latency in each subsystem is a difficult task in a modern server. Instead, we utilize the envelopes as a simple tool for controlling class quality-of-service while maximizing utilization of system resources.

Our approach has three key distinctions. First, it enables web servers to support a strong service model with class latencies bounded to a prespecified target, i.e., a minimum fraction of accepted requests will be serviced within the class delay target.

• *The authors are with the Department of Electrical and Computer Engineering, Rice University, Houston, TX 77005.*
E-mail: {kanodia,knightly}@rice.edu.

Manuscript received 29 Sept. 2000; revised 1 Nov. 2001; accepted 1 May 2002.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 112928.

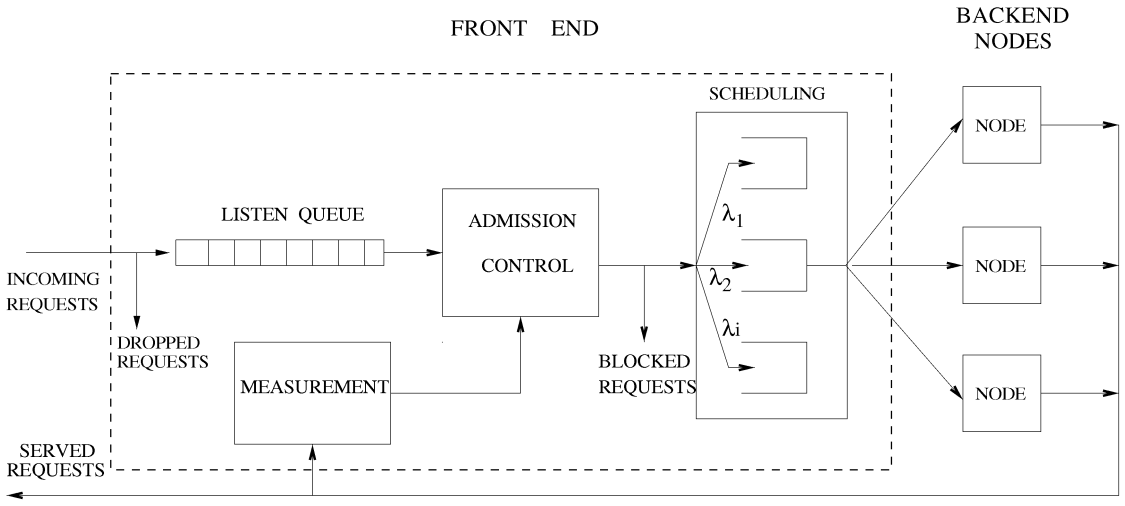


Fig. 1. System model.

Second, it provides a mechanism to characterize and control the interclass relationships. For example, suppose server resources are allocated to classes in a weighted-fair manner so that classes have performance isolation, yet a class is able to utilize unused resources of other classes. In general, the extent to which an increased load in one class affects the performance of another class is a complex function of the total system load, the particular resource scheduling algorithm, and the low-level interactions among the server's resources. Building on the results of [19], we use the envelopes as a way to characterize the high-level isolation/sharing relationships among classes and design a general multiclass algorithm to exploit these effects.

Finally, by decoupling access control and resource allocation from the internals of the server, we obtain a general solution that applies to a broad class of servers, including single-node and distributed servers, and servers with varying levels of quality-of-service support. Consequently, as the server is enhanced with functionalities such as weighted-fair resource allocation, the algorithm naturally exploits these features to better utilize the available resources and support an increased number of sessions per service class.

To evaluate our scheme, we perform a broad set of trace-driven-simulation experiments. We first compare our scheme with an uncontrolled system and illustrate that the algorithm is able to prevent performance degradation due to overload. Next, comparing the delays obtained in simulations with the class QoS objectives, we find that, in many cases, latencies can be controlled to within several percent of the targeted value. Moreover, in the single class case, we compare with a simple queuing theoretic approach, and find that envelopes control the system to a significantly higher degree of accuracy. Finally, in the multiclass case, simulations indicate that substantial interclass resource sharing gains are available. Here, we find that the approach is able to extract these gains and efficiently utilize system resources while satisfying each class' delay targets.

The remainder of this paper is organized as follows: In Section 2, we describe the server architecture and the system abstraction used for QoS management. Next, in Section 3, we describe a simple single-class queuing theoretic approach to serve as a benchmark for performance analysis, and illustration of the key problems in meeting delay targets. In Section 4, we introduce the request and service envelopes and show how they can be computed. Next, we develop an access control algorithm based on the properties of these envelopes in Section 5. We describe the simulation scenario and present experimental results in Sections 6 and 7, respectively. Finally, in Section 8, we conclude.

2 System Architecture

In this section, we describe the basic system architecture of a QoS web server. We are not proposing a new architecture as all of the mechanisms described below have been introduced previously. Rather, our goal is to consider a general system model for admission control which can exploit various QoS server mechanisms to efficiently satisfy targeted class latency objectives.

Fig. 1 depicts the general system model that we consider. The system consists of a state-of-the-art web server augmented with admission control capabilities as in [4], [10], [16]. All incoming requests, which can be sessions (as in [10]) or individual "page" requests, are classified into different quality-of-service classes. There are a number of possible classification criteria including the address of the server (in case of Web hosting applications), the identity of the user issuing the request, or the particular application or data type. The goal of our admission control algorithm is to determine whether admission of a new request in a particular service class can be supported while meeting the latency targets of *all* classes. If it is not possible, the request should be rejected outright or redirected to a lower priority class or a different server.

As shown in Fig. 1, incoming requests are first queued onto the listen queue or dropped if the listen queue is full. The admission controller dequeues requests from the listen queue and determines if they will be admitted or rejected. Notice that the admission control unit is part of the front-

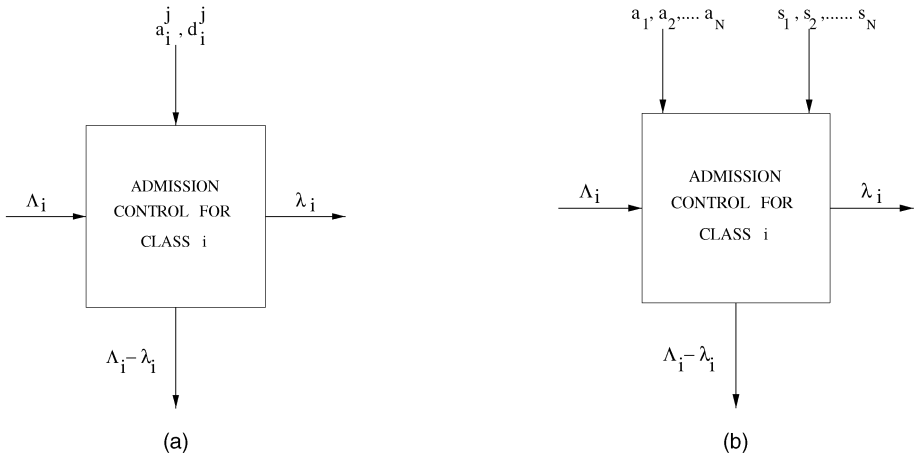


Fig. 2. Admission control model. (a) Baseline algorithm and (b) LMAC.

end and monitors all of the server's arrivals and departures. As depicted in Fig. 2,[1] the admission control unit performs observation-based control of the server using measured request and service rates of each class. Further, notice from Fig. 2b that our class-based admission control will also consider the effects of a new admission on *other* service classes.

If admitted, requests are then scheduled according to the server's request scheduling algorithm which can be first-come-first-serve or class based [4], [12]. Finally, requests are submitted to the back-end nodes in a distributed server [2], or in the case of a single node server are simply processed by the node itself.

A key point is that the admission controller applies to a general system model including single-node and distributed servers, FCFS and class-based scheduling, and standard as well as QoS-enhanced operating systems. When QoS mechanisms are present in the server (such as class-based rather than FCFS request scheduling), the admission controller will measure the corresponding performance improvements and exploit the QoS functionality by admitting more requests per class,[2] thereby increasing the overall system efficiency. For example, consider a server farm where the front-end does sophisticated load balancing to achieve better overall throughput by exploiting locality information at the back end [18]. In this case, the admission controller will measure the decreased service latencies and be able to admit an increased number of sessions into various classes, thereby exploiting the efficiency gain of load balancing. Finally, notice from Fig. 2 that the admission controller does not measure or model resources at the operating system level, such as disk, memory, or CPU. Instead, we abstract all low-level resources into a virtual server which allows us to design an admission controller that is applicable to a broad class of web server architectures and applications.

1. We refer the reader to Table 1 for the notation used in this figure.

2. The increase in number of admitted requests is as compared with an admission controller running on top of a server with no inbuilt QoS functionality.

3 Baseline Scheme

In this section, we sketch a simple queuing theoretic algorithm devised to satisfy a delay target. The goal here is threefold. First, we illustrate an abstraction of the server resources into a simple queuing model. Second, we highlight key issues for managing multiclass web services. Finally, we use the approach as a baseline for experimental comparisons and, by highlighting its limitations, we further motivate the LMAC scheme.

3.1 Problem Formulation

Consider a single class with quality-of-service targets given by a delay bound of 0.5 seconds to be met by 95 percent of requests. Further consider a stationary and homogeneous arrival of sessions and requests within sessions, so that there exists some maximum number of requests per second which can be serviced so that this QoS requirement is met. If the overall arrival rate of requests to the server is greater than this maximum, the difference should be blocked [3] (or redirected) by the access controller to prevent an overload situation.

The key question is, how to determine which load level is the maximum one that can support the service. Specifically, if the current load is below this maximum, then the current 95 percentile delay will be below the target. However, when a new session requests access to the server, the new 95 percentile delay of this class and others is, in general, a complex function of the system workload, and the low-level interactions among the many resources consumed such as disk, bandwidth, memory, and CPU. Below, we sketch a baseline approach for assessing the impact of new requests and sessions on the delay target via a simple queuing theoretic abstraction.

3.2 Sketch Algorithm

Here, we approximate class i's service by an M/M/1 queue with an unknown service rate. In particular, as described above, a request's service latency includes delays from session queues, disk access, etc. The M/M/1 model abstracts these resources into a single virtual server with

3. For the purposes of discussion here, we assume blocked requests to mean requests which are denied access to the server's resources.

independent and exponential requests and services as follows.

Over the last T seconds from the current time t, the mean arrival rate is

$$\lambda_i = \frac{\sum_j 1(t - T \leq a_i^j \leq t)}{T}, \quad (1)$$

where $1(\cdot)$ is an indicator function, and the mean delay is

$$\bar{d}_i = \frac{\sum_j (s_i^j - a_i^j) 1(t - T \leq s_i^j \leq t)}{\sum_j 1(t - T \leq s_i^j \leq t)}. \quad (2)$$

Under the assumptions of the M/M/1 model, the unknown service rate is simply

$$\mu_i = \frac{1}{\bar{d}_i} + \lambda_i, \quad (3)$$

so that the delay violation probability under an increased load $\lambda_i' > \lambda_i$ will be

$$P(D_i > d_i^*) = \exp\big(-d_i^*(\mu_i - \lambda_i')\big). \quad (4)$$

Thus, the increased load due to the new session should be admitted if the estimated $P(D_i > d_i^*)$ is less than the class' target ϵ_i^*. Consequently, under the particular assumptions of the M/M/1 model, the above scheme limits the class' latency to within the target d_i^* for the specified fraction of requests ϵ_i^*.

3.3 Limitations of the Baseline Scheme

While server access control based on (1)-(4) does provide the ability to meet a class' latency objectives with a high-level abstraction of system resources, it encounters several key problems which preclude its practicality to realistic web servers.

First, it offers no support for multiple services classes. That is, by treating each class independently, the impact of a new session on *other* classes is ignored. Second, the assumption that interrequest times are independent and exponentially distributed conflicts with measurement studies [11]. Third, the assumption of independent and exponentially distributed *service* times cannot account for the highly variable service times of requests and ignores the strong effects of caching, namely, that consecutive requests for the same document can result in highly correlated, as well as highly variable, service times.

In Section 7, we experimentally quantify the impact of these limitations in a realistic scenario.

4 Envelopes: A General Service and Demand Abstraction

In this section, we describe *envelopes* as a tool for developing an admission control algorithm which can overcome the limitations of the baseline scheme. Deterministic [13] and statistical [5], [9], [19] traffic envelopes have been developed to manage network QoS. Moreover, deterministic [13] and statistical [19] *service* envelopes have provided a foundation for multiclass network QoS, in [19], also incorporating inter-class resource sharing. Below, we extend these techniques to the scenario of measurement-based web services by exploiting two key properties of envelopes: characterization of temporal correlation and variance and simple online measurement via jumping windows.

4.1 Measurement-Based Request Envelopes

Here, we define and show how to measure a class' request envelope. We aim to characterize the traffic offered to a web server by measuring the mean and the variance of the incoming requests over intervals of length τ. The number of requests in an interval of length τ is $\lambda_i \tau$, where λ_i is as computed in (1).

Denoting the number of class i requests in the interval $[t_1, t_2]$ by (see Table 1 for the notation used)

$$N_i(t_1, t_2) = \sum_j 1\Big(t_1 < a_i^j < t_2\Big). \quad (5)$$

The variance of the request envelope, measured over a measurement window of length T, is given by [4]

$$\sigma_i^2(\tau) = \frac{1}{\lfloor T/\tau \rfloor - 2} \sum_{m=0}^{\lfloor T/\tau \rfloor - 1} \big(N_i(t - (m+1)\tau, t - m\tau) - (\tau \lambda_i)\big)^2. \quad (6)$$

As an example envelope, Fig. 3a shows the request envelope for the Rice University Computer Science Department trace described in Section 6. Specifically, the figure depicts $\lambda_i + 1.645\sigma_i(\tau)/\tau$ versus τ, where $1.645\sigma_i(\tau)$ yields the 95 percent tail of a Gaussian distribution. In other words, under a Gaussian distribution of total requests with empirical mean and variance as above, the figure shows the value of r such that $P(N_i(t - \tau, t)/\tau < r) = 0.95$. Fig. 3b shows the envelope normalized to the interval length so that the y-axis is a rate.

For example, in Fig. 3a, the point (100 msec, 17) on the curve indicates that 17 consecutive requests arrive within 100 msec 95 percent of the time. This corresponds to a rate of 170 requests per second over the same interval length which is depicted in Fig. 3b. Thus, the figure shows that over short interval lengths, significantly more requests than the mean 100 per second (as can be seen from the rate to which the curve in Fig. 3b converges) can arrive. Such characteristics of the request workload are a key input to admission control.

4.2 Measurement-Based Service Envelopes

Here, we define and show how to adaptively measure a class' service envelope. Analogous to the above request envelope, it describes the service latencies of consecutive requests which simultaneously compete for system resources, characterizing the variance and temporal correlation of services. In particular, we measure this envelope by monitoring the service latencies of requests as a function of the number of concurrent requests. For example, let k be the number of consecutive requests in consideration. For $k = 1$, the service envelope consists of the mean and variance of the time required to service a single request. For $k = 2$, the envelope characterizes the mean and variance of the time

4. The variance is the sum of variances over intervals of length τ. The variance for any interval is given by the square of difference between the sample point and the mean value.

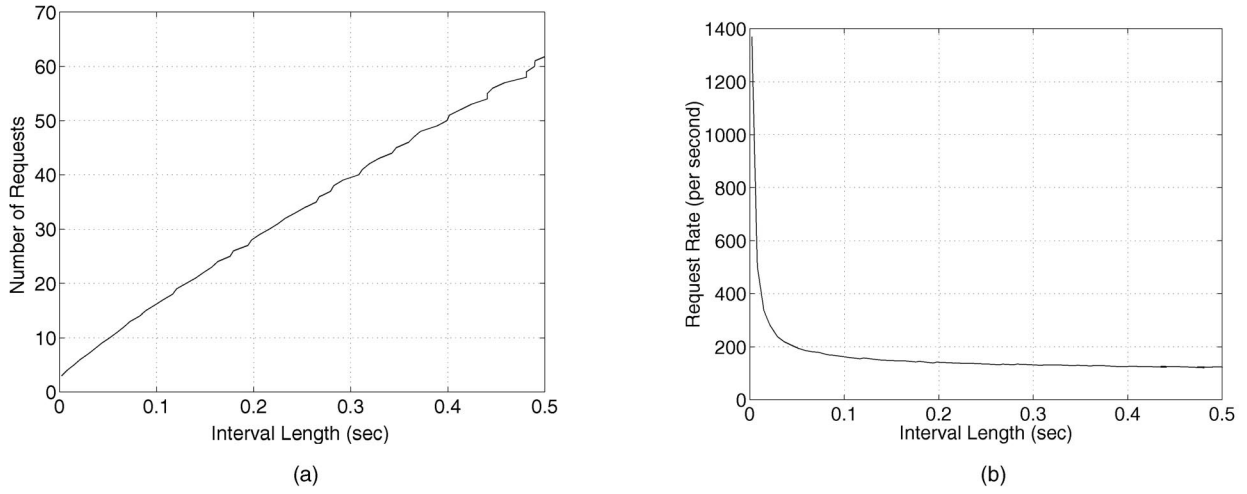


Fig. 3. 95 percentile request envelopes. (a) Cumulative envelope and (b) Rate envelope.

required to service two requests that are concurrently competing for system resources. That is, if the *j*th request enters the system before the $(j-1)$th request is serviced, then $(s_i^j - a_i^{j-1})$ represents the total time required to service the two requests.[5]

Thus, in general, we describe the service of class i by $d_i(k)$ and $\gamma_i^2(k)$ which are the mean and variance of the time to service k overlapping requests. Denoting $\beta_i^j(k)$ as indicator of whether request j overlaps with k (where $k \geq 2$) requests such that

$$\beta_i^j(k) = \begin{cases} 1 & s_i^{j+m} > a_i^{j+m+1},\ 0 \leq m \leq k-2 \\ 0 & otherwise, \end{cases} \quad (7)$$

then the mean latency to service k concurrent requests is given by[6]

$$\bar{d}_i(k) = \frac{\sum_j (s_i^j - a_i^{j-k+1}) \beta_i^j(k) 1(t - T \leq s_i^j \leq t)}{\sum_j \beta_i^j(k) 1(t - T \leq s_i^j \leq t)}. \quad (8)$$

Notice that $\bar{d}_i(1) = \bar{d}_i$ as in (2). Similarly, the service variance to serve k concurrent requests is given by

$$\gamma_i^2(k) = \frac{\sum_j (s_i^j - a_i^{j-k+1} - \bar{d}_i(k))^2 \beta_i^j(k) 1(t - T \leq s_i^j \leq t)}{(\sum_j \beta_i^j(k) 1(t - T \leq s_i^j \leq t)) - 1}. \quad (9)$$

Fig. 4 depicts an example service envelope from the simulation experiments of Section 7. Analogous to Fig. 3, it depicts the number of concurrent requests serviced as a function of the the latency incurred. The key property of the figure is its convexity so that, for example, the time required to service n requests is far less than n times the time required to service one request. This is due to the effects of caching (requests for the same document within close temporal proximity experience significantly smaller latencies) and, more generally, the server's ability to efficiently service concurrent requests.

5. Note that if the *j*th request enters the system after the $(j-1)$th request is serviced, then this duration reflects the two request's interarrival time rather than the time to service two requests.

6. This equation is valid only if $\beta_i^j(k)$ is not equal to zero.

5 Multiclass Admission Control

In this section, we build on the previous admission control model and introduce the Latency-Targeted Multiclass Admission Control (LMAC) algorithm. The goal is to provide a strong service model for Web classes that controls statistical latency targets of multiple service classes. The LMAC algorithm has two key distinctions from the baseline scheme of Section 3. First, we use envelopes as a general way of describing a class' service and demand. As for the baseline scheme, this is a high-level workload and service characterization, yet, unlike the baseline scheme, envelopes capture effects of temporal correlation and high variability in requests and service latencies. Second, exploiting the interclass theory of [19], we show how the performance effects of one class on another can be incorporated into admission control decisions.

5.1 Sketch LMAC Algorithm

The LMAC test is invoked upon arrival of a new session or request in class i which will increase the request rate from

TABLE 1
Notation

a_i^j	arrival time of admitted request j in class i
s_i^j	service time of request j in class i
$\bar{d}_i$	mean delay of class i requests
λ_i	mean arrival rate of admitted class i requests
μ_i	mean service rate of class i requests
d_i^*	class i target delay
ϵ_i^*	class i delay-violation probability
$\sigma_i^2(\tau)$	variance of class i requests over durations τ
$\bar{d}_i(k)$	mean class i latency for k concurrent requests
$\gamma_i^2(k)$	variance of class i latency for k concurrent requests
T	measurement window

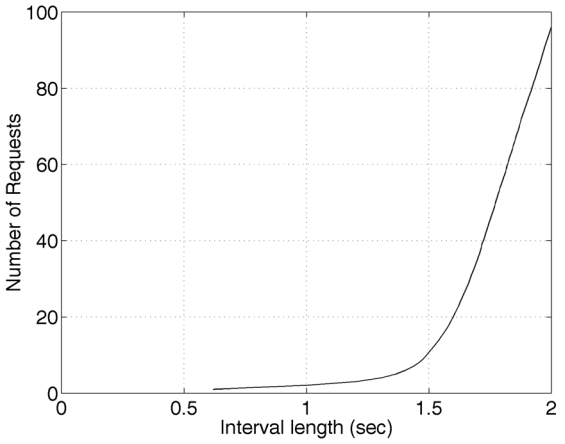


Fig. 4. Service envelope.

its current value λ_i to $\lambda_i' > \lambda_i$. The LMAC test consists of two parts: the first ensures class i's delay target is satisfied and the second ensures that other classes will not suffer QoS violations due to the increased workload of class i. We illustrate the test pictorially using Fig. 3 and Fig. 4.

For class i itself, with a statistical characterization of both requests and service, maintaining a maximum horizontal distance of d_i^* between the two curves ensures that the delay target is satisfied with probability ϵ_i (see [19], [15] for further details). With an increase in λ_i, class i itself increases its request rate yet retains its previous service level. Hence, the latency target is satisfied if the two curves remain d_i^* apart after an increase of $(\lambda_i' - \lambda_i)\tau$ in the request envelope.

For classes $l \neq i$, we must also consider the performance effects of class i on class l's delay targets. Our approach is to bound the effects of the incremental load resulting from the arrival of the new request. Specifically, by an upper bound on class l's new latency, we ensure that class i does not force class l into QoS violations. Moreover, by applying this bound only to the *incremental*[7] load the performance penalty for this worst-case approach is mitigated. This is true as we expect the load currently being serviced by the server to be large as compared with the load currently under service. In other words, class l's current service measurements incorporate the effects of class i's load λ_i, so that only $\lambda_i' - \lambda_i$ is included in the bound. The bound is obtained by considering that the incremental requests $\lambda_i' - \lambda_i$ have strict priority over class l sessions. Under this worst-case scenario, class l's workload remains the same yet its service over intervals of length τ is decreased by $(\tau + d_i^*)(\lambda_i' - \lambda_i)$.[8] Hence, the new request can be admitted if each class $l \neq i$ can satisfy its d_l^*, even under a reduction in service by $(\tau + d_i^*)(\lambda_i' - \lambda_i)$.

We make three observations about the LMAC test. First, each class' service envelope captures the gains from interclass resource sharing. For example, if class i can exploit unused capacity from an idle or lower priority class l, class i measures a correspondingly larger service envelope. Similarly, if the server has complete isolation of classes (e.g., via separate back-end nodes in the extreme case), then no such gains will be available and the algorithm will correctly limit admissions to reflect this. Second, the algorithm also ensures class performance isolation, i.e., that admissions in one class do not cause violations in another class, by incorporating the effects of interclass interference. Finally, we note that, while LMAC attempts to maximize the utilization of the Web server subject to the QoS constraints independent of the server architecture, QoS functionality in the server itself remains critical. For example, if a Web server provides no QoS support and no class differentiation, LMAC infers that only a single service can truly be provided, and restricts admissions to satisfy the most stringent class requirements. Alternatively, when QoS mechanisms are deployed in the Web server [2], [3], the resulting efficiency gains are in turn exploited by the LMAC algorithm, which increases the number of admitted sessions in each class and hence the overall system utilization.

7. The new session/request being considered for admission.

8. This follows from the fact that if the incremental load has static priority over class l sessions, then the incremental load gets service *before* any of class l's sessions.

6 Simulation Scenario

Our simulation scenario consists of a prototype implementation of the LMAC algorithm built into the simulator described in [18], which was developed to approximate the behavior of OS management for CPU, memory, and caching/disk storage. The front end node has a listen queue in which all incoming requests are queued before being serviced. Each back-end node consists of a CPU and locally attached disk(s) with separate queues for each. In addition, each node maintains its own main memory cache of configurable size and replacement policy.

Upon arrival, each request is queued onto the listen queue or dropped if the listen queue is full. Processing a request requires the following steps: dequeuing from the listen queue, connection establishment, disk reads (if needed), data transmission, and, finally, connection tear down. The processing occurs as a sequence of CPU and I/O bursts. The CPU and I/O bursts of different requests can be overlapped but the individual processing steps for each request must be performed in sequence. Also, data transmission immediately follows disk read for each block.

We have used the same costs for basic request processing as in [18]. The numbers were derived by performing measurements on a 300 MHz Pentium II machine running FreeBSD 2.2.5 and an aggressive experimental server.

Connection establishment and tear down costs are set to 145 μs of CPU time each. Transmit processing costs are 40 μs of CPU time per 512 bytes. Reading a file from the disk requires a latency of 28 ms for two seeks plus rotational latency followed by a transfer time of 410 μs per four KByte (resulting in a peak transfer rate of 10 MBytes/sec). A file larger than 44 KBytes is charged an additional latency of 14 ms (one seek plus rotational latency) for every 44 KByte block length in excess of 44 KBytes. The cache replacement policy is Greedy-Dual-Size [8]. To incorporate cache behavior, we deliberately set the cache size in our simulation to be 32 MB. The small cache size effectively compensates for the relatively small data set of our traces since only a subset of requested files can now be cached.

The input to the simulator are streams of tokenized requests, one stream for each user class. Requests within a user class arrive with a user-defined mean rate. Each request represents a file (and the corresponding file size in bytes). We generate the arrival stream from logs collected from real Web servers.

The latency experienced by a request is the delay from the time the request arrives at the listen queue until the time when that connection is torn down. The time taken to make admission control decisions are assumed to be negligible. This does not imply that we completely ignore the effect of the server time spent in processing eventually rejected requests. In fact, as can be seen from Fig. 1, all admission control decisions are made after the server dequeues a request from the listen queue. Thus, any rejected requests have used up some of the resources of the server (namely, the time spent in the finite sized listen queue). We ignore the actual processing time spent while making the admission control decision. Also, while calculating the arrival envelope we assume the incoming request is of the average size. Once accepted, the service time of the request is a function of its size which is taken into account while calculating the service envelope.

6.1 Web Server Traces

The input to the simulator is derived from Web server logs of two Web servers, 1) the Web server of the Computer Science Department at Rice University and 2) the Web server for the 1998 World Cup Soccer [1]. The Rice University trace contains requests for 26,947 unique files with a data set of 918 MB and an average request size of 28 KB. The trace does not contain the request arrival times. For simplicity, we simulate interarrival times as exponential. We vary the load on the server (in terms of average reqs/sec) by varying the interarrival times of the requests, keeping the arrival distribution exponential.

The other trace used as input to the simulator is a subset of the trace derived from the 1998 World Cup Soccer web server. The trace contains requests for 5,200 unique files with a data set of approximately 91 MB and an average request size of seven KB. To vary the load upon the server and also to reduce experimental runtime, we modified the request time stamps in the original trace in such a way that the load was scaled by different factors, thus, keeping the order of the requests and the relative interarrival times between different requests the same.[9] While deriving the trace for our experimental investigations, we ignored the requests with HTTP method HEAD and POST. POST requests represent CGI requests and as we have no way of duplicating the processing time involved, we ignore these requests. The POST and HEAD requests constituted approximately 0.55 percent of all requests.

7 Experimental Investigations

In this section, we describe the experiments performed to investigate the performance of the LMAC algorithm. The first experiment was performed to demonstrate LMAC's capability to actually protect the server from overload. In particular, without admission control, as the load offered to a server is increased beyond the server's capacity, the request latencies become excessive. Admission control provides protection from overload by monitoring the utilization of the server and blocking requests which will yield unacceptable performance.

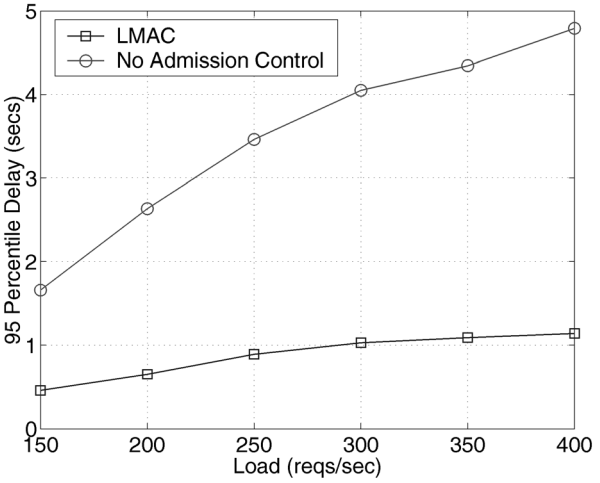


Fig. 5. 95 percentile latency versus load for Rice University computer science trace.

7.1 Overload Protection

To demonstrate the overload protection capabilities of the LMAC algorithm, we simulate various offered loads to the Web server, keeping the targeted request latencies to be the same. The trace used was the Rice University Computer Science Departmental trace. We compare the performance of LMAC with a Web server without admission control capabilities and measure the 95 percentile delay in both cases. Fig. 5 shows the results for a targeted delay of one second. As depicted in the figure, latencies in the unmodified server increase without bound as the load is increased. On the other hand, the LMAC algorithm blocks requests to meet the delay constraint and thus protects the web server from overload. In addition to overload protection, the figure indicates that LMAC controls latencies to within a small error of the target.

7.2 Accuracy

For the second experiment, we compare the performance of LMAC with the queuing theoretic baseline approach of Section 3. (We do not compare with admission control schemes from the literature as none have latency targets.) Fig. 6 shows the measured throughput versus 95 percentile delay for an offered load of 200 requests/sec, for the Rice University computer science trace. The server is a stand alone server with all incoming requests belonging to a single user class. The Baseline and the LMAC curves depict the throughput obtained when targeting 95 percentile delay values of 0.25, 0.5, 1, 1.25, 1.5, 1.75, 2, and 2.5 seconds. The Simulation curve depicts the measured value of 95 percentile delay and measured value of throughput when targeting the 95 percentile delay values as given above.[10]

9. Note that, the scaling does not change the distribution of the arrival process, only the mean and variance.

10. Note that, for any given targeted value of 95 percentile delay, both the baseline and the LMAC scheme achieve different values of 95 percentile delay at different throughput values.

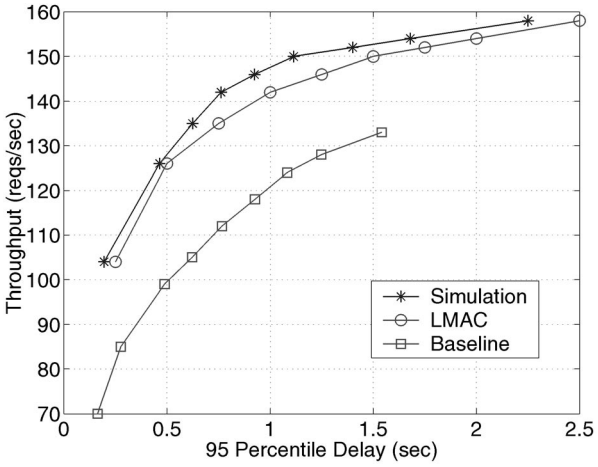


Fig. 6. Throughput versus 95 percentile latency.

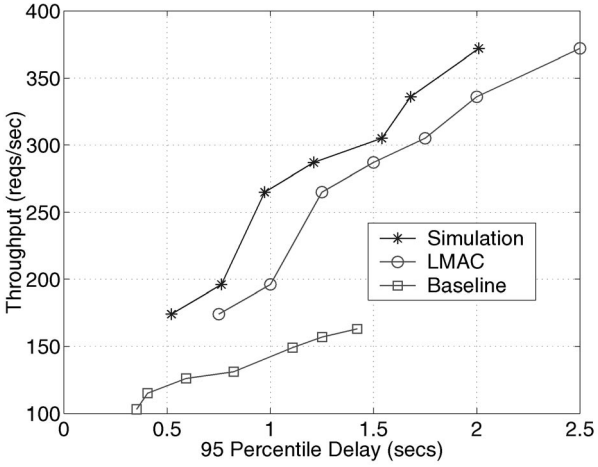


Fig. 7. Throughput versus 95 percentile latency for the World Cup Soccer trace.

Both the LMAC algorithm and the baseline approach meet the latency targets, yet the baseline scheme blocks an excessive number of requests thereby unnecessarily restricting throughput. The low utilization level of the baseline approach can be explained by the fact that the assumption of independent and exponentially distributed service and arrival times does not take into account the inherent variability introduced by Web traces and the Web server itself. For example, back-to-back requests for the same document result in lower delays for subsequent requests (since the document will reside in cache), yet the baseline approach does not exploit this correlation when performing admission control. On the other hand, the LMAC algorithm incorporates temporal correlation and variance properties of requests and services and achieves a correspondingly higher throughput. Regardless, LMAC is still somewhat conservative. For example, for a targeted 95 percentile latency of one second, a 95 percentile latency of 0.76 seconds is measured (the Simulation curve in Fig. 6) at a throughput of 141 reqs/sec. This means that when we perform simulations targeting a 95 percentile latency of one second, actual measurements give a 95 percentile latency of approximately 0.76 seconds, by blocking some of the incoming requests. But, LMAC could have allowed a larger number of requests into the system while still maintaining the targeted latency value. This is illustrated by the fact that for the Simulation curve in Fig. 6, the 95 percentile latency value of one second occurs at a sustainable throughput of 147 reqs/sec. Nevertheless LMAC does manage to meet latency targets with a utilization significantly higher than that of the baseline case.

Fig. 7 shows the results of a similar set of experiments with the input trace being the trace derived from the World Cup Soccer trace. The offered load on the server in this case is 400 reqs/sec while targeting 95 percentile delay values of 0.75, 1.0, 1.25, 1.50, 1.75, 2.0, and 2.5 seconds. From the figure, we observe that LMAC again satisfies the QoS targets and outperforms the baseline scheme. However, LMAC is more conservative than with the previous trace as indicated by the larger distance between the curves titled Simulation and LMAC. This difference arises from a number of aspects in the workloads such as the number of files, mean file size, request interarrival times, and request distribution (which impacts caching).

7.3 Multiclass Experiments

To investigate the performance of LMAC in a multiclass environment, we simulate a two-class scenario by randomly classifying incoming requests as belonging to one of the two classes, with each class having a different arrival rate and latency target.

An important point to note here is that a Web server without QoS capabilities would only be able to provide a single level of service. Hence, if two differentiated classes are targeted by admission control, the resulting request latencies will be those of the class with the minimum targeted latency; indeed, this behavior was confirmed by our experiments with the simulator.

In order to explore a true multiclass scenario, we devise an artificial resource isolation policy. We consider a server with two back-end nodes and a front-end policy in which the scheme of [18] is modified so that class 1 requests can be directed to either back-end node but all class 2 requests are directed only to one particular back end node. Thus, class 1 receives a minimum of one node's resources yet is able to exploit unused resources of node 2, whereas class 2 receives a *maximum* of one node's resources. While further class differentiation can be provided by additional QoS server mechanisms described in Section 1, this scenario allows a basic exploration of multiclass issues.

We perform two experiments. First, we perform simulations with complete isolation of the two classes (all class 1 jobs are directed to node one and all class 2 jobs are directed to node 2) so that there is no interclass resource sharing. Next, we perform the experiment as described above so that class 1 exploits interclass resource sharing. The results of the experiments are shown in Table 2. The input trace used for the results shown in Table 2 was the Rice computer Science trace. We obtained similar results with the World Cup soccer trace.

The request rate for class 1 is 300 reqs/sec with a delay target of 0.5 seconds and for class two the request rate is 200 reqs/sec with a delay target of one second. Observe

TABLE 2
Multiclass Performance of LMAC for the Rice Computer Science Trace

Class	Isolation		Multi-class with Sharing	
	Throughput (reqs/sec)	Delay (95 percentile)	Throughput (reqs/sec)	Delay (95 percentile)
1	147	.467	141	.501
2	92	.912	145	.935

from Table 2 that when isolated, both classes meet their delay targets at different throughput values, as obtained by the LMAC algorithm. More importantly, when the back-end nodes perform load balancing, the system itself is providing interclass resource sharing and the LMAC algorithm exploits these gains. Specifically, with the above load-balancing scheme, class two's throughput increases significantly while both classes' delay targets remain satisfied leading to a net higher level of server utilization. Thus, as described in Section 5, LMAC can satisfy an arbitrary set of class QoS targets, yet its efficiency in doing so relies in the QoS functionality of the server itself. Regardless, the goal of LMAC is to maximally utilize system resources, given whatever QoS targets are required, and whatever server QoS mechanisms are present.

8 Conclusions

In this paper, we developed a scheme termed Latency-targeted Multiclass Admission Control (LMAC). The algorithm uses measurements of requests and service latencies to control each class' quality-of-service. By abstracting system resources into a high-level virtual server rather than modeling the intricate interactions of low-level system resources, our approach can be applied to off-the-shelf servers enhanced with monitoring and admission control. Moreover, as QoS and performance functionalities are added to servers, e.g., class-based request scheduling, operating systems enhanced with quality-of-service mechanisms, or locality aware load balancing, we have shown that the LMAC algorithm exploits these features and realizes a corresponding increase in utilization to various service classes.

In future work, we plan to experiment with additional traces. Further, since dynamic content is becoming a major component in Web server workloads, we plan to extend LMAC to take into account dynamic content. We also plan to investigate the performance of LMAC in presence of persistent connections. We are also investigating the performance of LMAC and similar measurement-based admission control approaches in large-scale Internet Data Centers (IDC) [20].

Acknowledgments

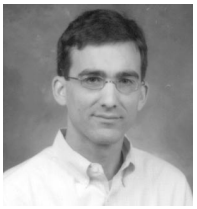
The authors are grateful to Mohit Aron, Peter Druschel, Vivek Pai, and Willy Zwaenepoel for helpful discussions and assistance with the simulator and traces. This paper was presented in part at the Eighth IEEE/IFIP International Workshop on Quality of Service, Pittsburgh, June 2000 [14]. This research is supported by the US National Science Foundation Grants ANI-9733610 and ANI-0085842, and by a Sloan Fellowship.

References

[1] M. Arlitt and T. Jin, "Workload Characterization of the 1998 World Cup Web Site," *IEEE Network,* vol. 14, no. 3, pp. 30-37, May 2000.
[2] M. Aron, P. Druschel, and W. Zwaenepoel, "Cluster Reserves: A Mechanism for Resource Management in Cluster-Based Network Servers," *Proc. ACM SIGMETRICS 2000,* June 2000.
[3] G. Banga, P. Druschel, and J. Mogul, "Resource Containers: A New Facility for Resource Management in Server Systems," *Proc. Third USENIX Symp. Operating Systems Design and Implementation,* Feb. 1999.
[4] N. Bhatti and R. Friedrich, "Web Server Support for Tiered Services," *IEEE Network,* vol. 13, no. 5, pp. 64-71, Sept. 1999.
[5] R. Boorstyn, A. Burchard, J. Liebeherr, and C. Oottamakorn, "Effective Envelopes: Statistical Bounds on Multiplexed Traffic in Packet Networks," *Proc. IEEE INFOCOM 2000,* Mar. 2000.
[6] J. Bruno, J. Brustoloni, E. Gabber, B. Ozden, and A. Silberschatz, "Retrofitting Quality of Service Into a Time-Sharing Operating System," *Proc. 1999 USENIX Ann. Technical Conf.,* 1999.
[7] J. Bruno, E. Gabber, B. Ozden, and A. Silberschatz, "The Eclipse Operating System: Providing Quality of Service Via Reservation Domains," *Proc. 1998 USENIX Ann. Technical Conf.,* 1998.
[8] P. Cao and S. Irani, "Cost Aware WWW Proxy Caching Algorithms," *Proc. USENIX Symp. Internet Technologies and Systems (USITS),* Dec. 1997.
[9] C. Cetinkaya, V. Kanodia, and E. Knightly, "Scalable Services Via Egress Admission Control," *Proc. IEEE/ACM Trans. Multimedia: Special Issue on Multimedia over IP,* vol. 3, no. 1, Mar. 2001.
[10] L. Cherkasova and P. Phaal, "Session-Based Admission Control: A Mechanism for Improving Performance of Commercial Web Servers," *Proc. IEEE/IFIP Int'l Workshop Quality-of-Service '99,* June 1999.
[11] M. Crovella and A. Bestavros, "Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes," *Proc. IEEE/ACM Trans. Networking,* vol. 5, no. 6, pp. 835-846, Dec. 1997.
[12] M. Crovella, R. Frangioso, and M. Harchol-Balter, "Connection Scheduling in Web Servers," *Proc. 1999 USENIX Symp. Internet Technologies and Systems (USITS '99),* Oct. 1999.
[13] R. Cruz, "Quality of Service Guarantees in Virtual Circuit Switched Networks," *Proc. IEEE J. Selected Areas in Communications,* vol. 13, no. 6, pp. 1048-1056, Aug. 1995.
[14] V. Kanodia and E. Knightly, "Multiclass Latency-Bounded Web Services," *Proc. IEEE/IFIP Int'l Workshop Quality-of-Service 2000,* June 2000.
[15] E. Knightly and N. Shroff, "Admission Control for Statistical QoS: Theory and Practice," *IEEE Network,* vol. 13, no. 2, pp. 20-29, Mar. 1999.
[16] K. Li and S. Jamin, "A Measurement-Based Admission Controlled Web Server," *Proc. IEEE INFOCOM 2000,* Mar. 2000.
[17] J. Mogul, "Operating System Support for Busy Internet Servers," *Proc. Fifth Workshop on Hot Topics in Operating Systems,* May. 1995.
[18] V. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum, "Locality-Aware Request Distribution in Cluster-Based Network Servers," *Proc. Eighth Conf. Architectural Support for Programming Languages and Operating Systems,* Oct. 1998.
[19] J. Qiu and E. Knightly, "Interclass Resource Sharing Using Statistical Service Envelopes," *Proc. IEEE INFOCOM '99,* Mar. 1999.

[20] S. Ranjan, J. Rolia, H. Fu, and E. Knightly, "QoS-Driven Server Migration for Internet Data Centers," *Proc. IEEE/IFIP Int'l Workshop Quality-of-Service 2002,* May 2002.

Vikram Kanodia received the BTech degree in electronics and electrical communications engineering from the Indian Institute of Technology, Kharagpur, India, in 1998 and the MS degree in electrical and computer engineering from Rice University, Houston, Texas, in 2000. He is a PhD candidate at Rice University. Currently, he is working with the Rice Networks Group at Rice University. His current research interests include quality of service in wired and wireless networks, Web quality of service, and network management.



Edward Knightly received the BS degree from Auburn University in 1991, the MS degree from the University of California at Berkeley in 1992, and the PhD degree from the University of California at Berkeley in 1996, all in electrical engineering. Since 1996, he has been an assistant professor in the Department of Electrical and Computer Engineering at Rice University, Houston, Texas. He currently serves on the editorial board of the *Computer Networks Journal*, *IEEE/ACM Transactions on Networking*, and *IEEE Transactions on Multimedia*. He served as cochair for the 1998 International Workshop on Quality of Service and is the finance chair for ACM MOBICOM 2002. He received the US National Science Foundation CAREER award in 1997 and the Sloan Fellowship in 2001. His research interests are in the area of theory, algorithms, and architectures for quality of service in wired and wireless network.