A Fast and Accurate Iterative Solution of a Multi-class Threshold-based Queueing System with Hysteresis

Leana Golubchik*
Department of Computer Science & UMIACS
University of Maryland at College Park, USA
leana@cs.umd.edu

John C.S. Lui†
Department of Computer Science & Engineering
The Chinese University of Hong Kong
cslui@cse.cuhk.edu.hk

ABSTRACT

In this paper, we consider a K-server *multi-class* threshold-based queueing system with hysteresis in which the number of servers, employed for servicing customers of each class i, is governed by a *forward threshold* vector $\boldsymbol{F}_i = [F_i(1), F_i(2), \ldots, F_i(K_i - 1)]$ and a *reverse threshold* vector $\boldsymbol{R} = [R_i(1), R_i(2), \ldots, R_i(K_i - 1)]$. There are many applications and systems where a multi-class threshold-based queueing system can be of great use. One motivation for using threshold-based techniques is that such systems incur significant server setup, usage, and removal costs. And, as in most practical situations, an important concern is not only the system performance but rather its cost/performance ratio. The motivation for use of hysteresis is to control the cost during momentary fluctuations in workload. Moreover, servers in such systems are often needed by multiple classes of workloads, and hence, it is desirable to find good approaches to sharing these servers among the different workloads, preferably without statically partitioning the server pool among the classes; threshold-based techniques constitute one category of such approaches. Consequently, an important and distinguishing characteristic of our work is that we consider a *multi-class* system, which is needed in modeling of many applications and systems. Our main goal in this work is to develop an efficient method for solving such models and computing the corresponding performance measures of interest, which can subsequently be used in evaluating designs of threshold-based systems.

1. INTRODUCTION

In this paper, we consider a *multi-class* K-server threshold-based queueing system with hysteresis in which the number of servers, employed for servicing customers of class i, $i = 1, \ldots N$, is governed by a *forward threshold* vector $\boldsymbol{F}_i = [F_i(1), F_i(2), \ldots, F_i(K_i - 1)]$ (where $F_i(1) < F_i(2) < \cdots < F_i(K_i - 1)$) and a *reverse threshold* vector $\boldsymbol{R}_i = [R_i(1), R_i(2), \ldots, R_i(K_i - 1)]$ (where $R_i(1) < R_i(2) < \cdots < R_i(K_i - 1)$). This multi-server multi-class queueing system has a total of K servers where the allocation of servers to classes is performed as follows. Each class is allocated a minimum of one server. Thus, a customer of class i arriving to an empty system is served by a single server. A new arrival of a class i customer to a system with $F_i(j)$ class i customers already there (and j servers already allocated to that class) forces an attempt to allocate one additional server to class i, where $j = 1 \ldots K_i - 1$, and K_i is the maximum number of servers that can be allocated for service of class i customers. A departure of a class i customer from a system (with $j+1$ servers allocated to that class prior to this departure) which leaves behind $R_i(j)$ customers of class i forces a de-allocation of a server, where $j = 1 \ldots K_i - 1$ — that is, it forces the return of a server, that was earlier allocated to class i, to the pool of "free" servers which are available for allocation to all classes. Hence, the N classes share a pool of K servers, with a *dynamic* allocation of servers to classes governed by a set of thresholds with hysteresis behavior. If $\sum_{i=1}^{N} K_i \leq K$, then the classes do not "interfere" with each other. Of course, the more interesting case is where $\sum_{i=1}^{N} K_i > K$, as motivated below.

One motivation for using a threshold-based approach is that many systems incur significant server setup, usage, and removal costs. And, as in most practical situations, an important concern of a system designer is not only the system performance but rather its cost/performance ratio. Furthermore, servers (resources) in a system are often needed by multiple classes of workloads (applications), and hence, it is desirable to find good approaches to sharing these servers among the different workloads, preferably without statically partitioning the server pool among the classes. More specifically, under light class i loads, it is not desirable to operate unnecessarily many servers for that class, due to the incurred setup and usage costs as well as due to the performance consequences of that class under-utilizing the servers while other classes are (possibly) experiencing high loads. On the other hand, it is also not desirable for a system to exhibit very long delays, which can result from lack of servers under heavy loads. One approach to improving the cost/performance ratio of a system is to *dynamically* react to changes in workload through the use of thresholds. For in-

*This work was supported in part by the NSF CAREER grant CCR-98-96232.

†This work was supported in part by the Mainline and RGC research grants.

stance, one can maintain the expected job response time in a system at an acceptable level, and at the same time maintain an acceptable cost for operating that system, by dynamically adding or removing servers, depending on the system load. Similarly, one can use the threshold-based server allocation approach to reduce the sensitivity of performance characteristics of a class of customers to the workload of other classes without having to statically partition resources between the classes. (We illustrate this further in Section 4.)

There are many applications where threshold-based resource management policies can be employed, and thus performance evaluation of such systems through analysis of *multi-class* threshold-based queueing systems can be of great use. For instance, the Novell file server maintains a memory pool such that a fraction of it is used for communication buffers and a fraction is used for file buffers, where threshold-based policies are implemented in order to make decisions about when to increase the number of network buffers and when to decreases it; the threshold values are based on perceived packet losses due to increases in network traffic activity. Similarly, OS design has been moving towards maintaining a common buffer space pool that can be dynamically managed between the various I/O processes. Furthermore, in transport protocols of communication networks [12], several transport-layer connections are multiplexed onto a single network layer connection. Whenever the traffic exceeds a certain threshold in the network-layer connection, another network-layer connection can be created to service the incoming traffic from the transport layer. Using such a control mechanism, severe degradations in throughput and delay can be avoided; at the same time operation costs can be kept at an acceptable level. Another example application is a system providing information query service via the Internet. As the number of queries increases, the number of servers, needed to maintain certain (acceptable) system response time characteristics, is also increased. Since the cost of setting up server connections can be significant[1], the use of a threshold-based approach can result in a cost-controlled creation and deletion of these connection, according to the changes in the workload. Thus, the model presented in this paper and its efficient solution will be beneficial for many systems and applications.

As in the case of electronic circuits that are prone to oscillation effects, a "simple" threshold-based system may not suffice. In a computer system, one reason for avoiding oscillations are the above mentioned server setup and removal costs, i.e., oscillations coupled with non-negligible server setup and removal costs can result in a poor cost/performance ratio of a system. More specifically, it is desirable to add servers only when a system is moving towards a heavily loaded operation region, and it is desirable to remove servers only when a system is moving towards a lightly loaded operation region — it is not desirable to alter the number of servers during "momentary" changes in workload, i.e., during oscillations. Such oscillation regions can be avoided by adding a hysteresis to the system — hence the motivation for looking for efficient analysis techniques of threshold-based queueing systems with *hysteresis* behavior.

[1]For instance, it may be necessary to broadcast information about the newly added server to the already active servers in the system.

As already mentioned, a threshold-based queueing system with hysteresis is defined by the forward and the reverse threshold vectors (see Section 2 for details). The actual values, or rather what are "good" values for these vectors is a function of many factors, such as the characteristics of the server setup, usage, and removal costs, characteristics of the arrival process and the service rates, as well as the possible "interaction" between the different classes of workloads. Our main goal in this work is to develop an efficient method for solution of multi-class multi-server threshold-based queueing models with hysteresis and computation of corresponding performance measures of interest. The question of optimal values for the threshold vectors is, in general, a difficult problem and is outside the scope of this paper. We must point out, however, that efficient model solution techniques can be of great use in evaluating various parameter settings (such as the threshold values) and hence are needed for performance evaluation of systems that manage resources in a threshold-based manner. Such analytical models are especially useful at design time, when the speed of evaluation is key. Thus, we believe that our solution method, due to its efficiency, facilitates accessible experimentation techniques for investigating the "goodness" of various threshold-based designs and parameter settings (refer to Section 4 for numerical examples).

Given the above motivation for the use of threshold-based systems with hysteresis, in this paper we present an efficient technique for solving the corresponding analytical models and computing various performance measures of interest. We begin with a very brief survey of some of the existing literature on the threshold-based queueing problem. A two-server system is considered in [13], [14], and [20]. An approximate solution for solving a degenerate form of this problem (where all thresholds are set to zero) is presented in [6, 8]; an approximate solution for a system that employs (non-zero) thresholds is presented in [21] (but without hysteresis). In [7], the authors solve a multi-server threshold-based queueing system with hysteresis, using the Green's function method [5, 9, 10]. In [16] we give a solution of several forms of the multi-server threshold-based queueing system with hysteresis using stochastic complementation [17]. Lastly, techniques for computation of bounds for performance measures of multi-server threshold-based queueing systems with hysteresis and non-instantaneous server activation are given in [3].

In this work, we consider and solve a *multi-class* multi-server threshold-based queueing system with hysteresis. The *contributions* of this work are as follows. To the best of our knowledge, *none* of the works described above give an efficient analytical solution technique for analyzing *multi-class* threshold-based systems with hysteresis behavior. Since in many applications, such as the ones described above, *multiple* types of workloads "compete" for a pool of resources, we consider it an important and distinguishing characteristic of our work. Specifically, we present an iterative solution technique which solves the multi-class model by "breaking" it up into N single class models, "coupled" through a set of model parameters which capture the interaction between classes. As shown in Section 4, in most test cases, this iterative approach produces accurate results and allows for efficient computation of performance measures of interest.

Furthermore, we study the performance characteristics of threshold-based systems and show that proper choices of design parameters, such as threshold values, can produce significant improvements in system performance. Using this study, we illustrate the utility of our approach in evaluating designs of threshold-based systems, where "good" parameter settings constitute not only an important but a difficult problem. We believe that the efficiency and accuracy of our approach facilitates large-scale experimentation with parameter settings and subsequent performance evaluation studies of threshold-based designs of systems.

Finally, we note that a variety of iterative approaches have been used in the literature for construction of approximation techniques (e.g., refer to [2]). For instance, an iterative technique for a somewhat different control schemes for dynamic resource sharing between multiple classes is employed in [18, 19].

The remainder of this paper is organized as follows. In Section 2 we give a description of our model (with further details given in [4]). Section 3 describes our iterative approach to solving the multi-class model (with details of the derivation of an individual class model solution, utilized by the iterative approach, given in [4]). The goodness of this approach, i.e., its accuracy and utility in system design and evaluation, is discussed in Section 4 through the use of numerical results. Finally, our conclusions are given in Section 5.

2. SYSTEM MODEL

In this section, we describe our *multi-class* threshold-based queueing model with hysteresis behavior which has an infinite state space and can be defined as follows. There is a total of K servers in the system, where K is unrestricted. The service time requirements of a class i customer are exponentially distributed with parameter μ_i. The customer arrival process is Poisson with rate λ and probability α_i that an arriving customer is of class i, where $\sum_{i=1}^{N} \alpha_i = 1$ and $1 \leq i \leq N$. That is, we consider a *multi-class* system with N classes, where N is unrestricted. Addition and removal of servers for serving customers of class i is governed by the forward and the reverse threshold vectors $\boldsymbol{F}_i = [F_i(1), F_i(2), \cdots, F_i(K_i - 1)]$ and $\boldsymbol{R}_i = [R_i(1), R_i(2), \cdots, R_i(K_i - 1)]$ where $F_i(j) < F_i(j+1)$ for $1 \leq j \leq K_i - 2$, $R_i(j) < R_i(j+1)$ for $1 \leq j \leq K_i - 2$, and $R_i(j) < F_i(j)$ for $1 \leq j \leq K_i - 1$.

Given a pool of K servers where each server is able to serve a customer of any class, each class i starts out with one server and may attempt to obtain at most K_i servers. These servers are allocated for service of class i customers and returned to the pool of available servers based on the number of class i customers currently in the system (as stated more formally below). In general, $\sum_{i=1}^{N} K_i$ may be greater than, equal to, or less than K; although the more interesting case is where $\sum_{i=1}^{N} K_i > K$.

Given a K server N class threshold-based queueing system with hysteresis, we model it as a Markovian process $\mathcal{M}$ with the following state space $\mathcal{S}$:

$$\mathcal{S} = \{(n_1, s_1, n_2, s_2, \ldots, n_N, s_N) \mid n_i \geq 0, s_i \in \{1, 2, \ldots, K_i\}, \sum s_i \leq K, i = 1, \ldots, N\}$$

where n_i is the number of class i customers in the system and s_i is the number of servers currently allocated to class i. Upon an arrival of a class i customer, if $F_i(j) \leq n_i \leq F_i(j) + a_i^j$ where $a_i^j \geq 0$ and $j = s_i$, then the system attempts to allocate an additional server for service of class i customers, which is possible *only* if $\sum_{i=1}^{N} s_i < K$. Note that in a system where $\sum_{i=1}^{N} K_i > K$, it may not always be possible to allocate another server to class i upon arrival, since at that time all K servers may have already been allocated. In this case, the arriving class i customer joins the queue of class i requests *as long as* $F_i(j) \leq n_i < F_i(j) + a_i^j$ (where $a_i^j \geq 0$ and $j = s_i$). When $n_i = F_i(j) + a_i^j$, the arriving class i customer is rejected by the system (i.e., dropped) *if* there is no server available for allocation to class i (i.e., if $\sum s_i = K$). For *correctness*, we assume the following constraint on all a_i^j:

$$F_i(j) + a_i^j < F_i(j+1) + a_i^{j+1}$$

for $i = 1, 2, \ldots, N$ and $j = 1, 2, \ldots, K_i - 1$. We also assume that $a_i^{K_i} = \infty$; hence, we have no restrictions on queue length when the maximum number of servers that maybe needed by class i have been allocated (i.e., when $s_i = K_i$). The limitation on queue length when $s_i < K_i$ is motivated by system design considerations. That is, if the system reaches a point where its design dictates that another server be allocated for class i workload, but a server is not available (i.e., all K of the system's servers are already allocated), then it is reasonable to assume that the system is (at least) temporarily overloaded, partly due to sharing of resources with other classes. And, rejection or blocking of customers is a reasonable approach to dealing with overload conditions. Of course, a "real" system will also not have an infinite queue length, when the maximum number of servers (K_i) for class i has been allocated. In this case, we may either (1) use a finite queue length model (i.e., $a_i^{K_i}$ is finite) and study the system's performance under a given queue size limitation, or (2) allow an infinite queue length (i.e., $a_i^{K_i} = \infty$) and use the model to study queue length requirements of the corresponding system. Our solution methodology (refer to Section 3) allows for either type of a model, but for simplicity of exposition, in the remainder of the paper we will focus our discussion on the infinite queue version (i.e., where a_i^j is finite, for $j = 1, \ldots, K_i - 1$, and $a_i^{K_i} = \infty$)[2].

Formally, the transition structure of $\mathcal{M}$ is as follows. The transitions corresponding to arrivals are:

$$(n_1, s_1, \ldots, n_i, s_i, \ldots, n_N, s_N) \xrightarrow{\lambda\alpha_i} (n_1, s_1, \ldots, n_i + 1, s_i, \ldots, n_N, s_N) \quad \text{if } C_1 \quad (1)$$

$$(n_1, s_1, \ldots, n_i, s_i, \ldots, n_N, s_N) \xrightarrow{\lambda\alpha_i} (n_1, s_1, \ldots, n_i + 1, s_i + 1, \ldots, n_N, s_N) \quad \text{if } C_2 \quad (2)$$

[2]Note that, the server allocation/deallocation scheme described here does not preclude potential idling of servers, due to (a) requiring that each class is allocated at least one server and (b) allocation of servers to classes on arrivals only. Many other control schemes are possible and are subject of future work.

where C_1 is

$$C_1 = \Big((s_i < K_i) \wedge (n_i < F_i(s_i))\Big) \vee \Big(s_i = K_i\Big) \vee \Big((s_i < K_i) \wedge (\sum_{j=1}^{N} s_j = K) \wedge (F_i(s_i) \leq n_i < F_i(s_i) + a_i^{s_i})\Big)$$

and C_2 is

$$C_2 = \Big(s_i < K_i\Big) \wedge \Big(\sum_{j=1}^{N} s_i < K\Big) \wedge \Big(F_i(s_i) \leq n_i \leq F_i(s_i) + a_i^{s_i}\Big)$$

The transitions corresponding to departures are:

$$(n_1, s_1, \ldots, n_i, s_i, \ldots, n_N, s_N) \xrightarrow{s_i\mu_i} (n_1, s_1, \ldots, n_i - 1, s_i, \ldots, n_N, s_N) \quad \text{if } C_3 \quad (3)$$

$$(n_1, s_1, \ldots, n_i, s_i, \ldots, n_N, s_N) \xrightarrow{s_i\mu_i} (n_1, s_1, \ldots, n_i - 1, s_i - 1, \ldots, n_N, s_N) \quad \text{if } C_4 \quad (4)$$

where C_3 is

$$C_3 = \Big((n_i > 0) \wedge (s_i = 1)\Big) \vee \Big((n_i > 0) \wedge (n_i - 1 > R_i(s_i - 1)) \wedge (s_i > 1)\Big)$$

and C_4 is

$$C_4 = \Big((n_i > 0) \wedge (n_i - 1 = R_i(s_i - 1)) \wedge (s_i > 1)\Big)$$

A more detailed explanation of the derivation of conditions C_1 through C_4 is given in [4].

3. ITERATIVE METHOD FOR SOLUTION OF A MULTI-CLASS MODEL

In this section we describe our *iterative* approach to solving the model presented in Section 2. As described in Section 2, the corresponding Markov process, $\mathcal{M}$, is infinite (in multiple dimensions), and hence our choices for solution are to either (a) simulate $\mathcal{M}$, or (b) look for special structure, or (c) look for efficient approximation techniques. Since $\mathcal{M}$ appears to lack sufficient structure for an efficient exact solution, below we describe an approximate solution technique, using iteration. The use of an approximation is motivated by the desire to construct an efficient solution approach (and simulation can be significantly slower than analytical solutions).

3.1 Basic Approach

The basic approach that we pursue here is as follows. The original model $\mathcal{M}$ is *approximately* "broken up" into N single class Markovian models, $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_N$, which are "coupled" through a set of blocking probabilities (see Section 3.2 for a more detailed description of the $\mathcal{M}_i$'s). More specifically, the interaction between classes occurs when class i requires allocation of another server (due to the crossing of a forward threshold), and no servers are available in the system (i.e., all K servers have already been allocated) due to the workload of other classes. Hence, in general, there is a non-zero probability that class i, which has already activated s_i servers, is not able to add a server upon the forward threshold crossing. Let us refer to this as a "blocking" probability $\mathcal{P}_{i,s_i}$, which (approximately) captures this interaction between classes[3]. We now describe our iterative approach.

Let $\mathcal{M}_i^{(n)}$ be the Markovian process corresponding to the individual class i model at iteration n with a corresponding steady state probability vector $\tilde{\pi}_i^{(n)}$. The parameters of each $\mathcal{M}_i^{(n)}$ are computed as a function of blocking probabilities, $\boldsymbol{P}_i^{(n)} = \{\mathcal{P}_{i,1}^{(n)}, \mathcal{P}_{i,2}^{(n)}, \ldots, \mathcal{P}_{i,K_i-1}^{(n)}\}$, which are in turn computed as a function of the steady state probability vector, $\tilde{\pi}_i^{(n-1)}$, obtained during the previous iteration. (We give the details of the construction of $\mathcal{M}_i^{(n)}$ and the computation of $\tilde{\pi}_i^{(n)}$ below[4].) Then, an overview of our iterative approach is (a more detailed and formal description is given in Section 3.3):

1. construct $\mathcal{M}_1^{(0)}, \mathcal{M}_2^{(0)}, \ldots, \mathcal{M}_N^{(0)}$; set $n = 0$ (this is iteration 0);
2. solve $\mathcal{M}_1^{(n)}, \mathcal{M}_2^{(n)}, \ldots, \mathcal{M}_N^{(n)}$, i.e., compute the corresponding steady state probabilities to obtain $\tilde{\pi}_1^{(n)}$, $\tilde{\pi}_2^{(n)}$, $\ldots$, $\tilde{\pi}_N^{(n)}$; set $n = n + 1$;
3. use these steady state probabilities to compute $\boldsymbol{P}_1^{(n)}$, $\boldsymbol{P}_2^{(n)}$, $\ldots$, $\boldsymbol{P}_N^{(n)}$;
4. use these blocking probabilities to update the individual class models, i.e., construct $\mathcal{M}_1^{(n)}$, $\mathcal{M}_2^{(n)}$, $\ldots$, $\mathcal{M}_N^{(n)}$, where for each $i = 1, \ldots, N$, parameters of $\mathcal{M}_i^{(n)}$ are computed as functions of $\boldsymbol{P}_i^{(n)}$ (but not $\boldsymbol{P}_j^{(n)}$ where $j \neq i$);
5. continue the iterative process (i.e., go back to step 2) until the values of all $\boldsymbol{P}_i$'s converge.

3.2 Individual Class Model

Since our iterative approach involves solution of individual class models ($\mathcal{M}_i$'s) we now briefly describe the class i model, which can be defined as follows. There are K_i servers (K_i is unrestricted), each with an exponential service rate μ_i. Customer arrivals are governed by a Poisson process with rate $\lambda_i = \alpha_i\lambda$. Addition and removal of servers is governed by the forward and the reverse threshold vectors, namely $\boldsymbol{F}_i = [F_i(1), F_i(2), \ldots, F_i(K_i - 1)]$ and $\boldsymbol{R}_i = [R_i(1), R_i(2), \ldots, R_i(K - 1)]$.

Given a K_i-server *single* class threshold-based queueing system with hysteresis, we model it as a Markov process $\mathcal{M}_i$ with the following state space $\mathcal{S}_i$:

$$\mathcal{S}_i = \{(k, j) \mid k \geq 0, j \in \{1, 2, \ldots, K_i\}\}$$

where k is the number of customers in the class i queueing system and j is the number of allocated servers. Figure 1

[3]Of course, this is an approximation, and hence, the following description of the $\mathcal{M}_i$'s used in the iterative solution technique is also an *approximation*.

[4]Note that there are multiple approaches to constructing $\mathcal{M}_i^{(0)}$'s, i.e., multiple ways to start the iteration; we give details of one such approach below.

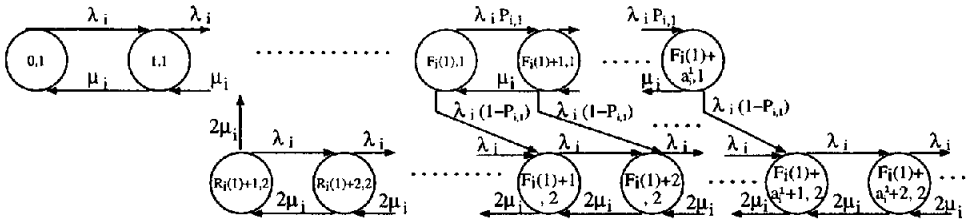

Figure 1: State transition diagram for a class i system with $K_i = 2$.

illustrates the state transition diagram for such a system where $K_i = 2$. Formally, the transition structure of $\mathcal{M}_i$ can be specified as follows[5], where all transitions are from state (k, j), with the state description given above:

Next State	Rate	Condition
$(k+1, j)$	λ_i	$(1 \le j < K_i) \wedge (k < F_i(j))$
$(k+1, j)$	λ_i	$j = K_i$
$(k+1, j)$	$\lambda_i \mathcal{P}_{i,j}$	$(1 \le j < K_i) \wedge (F_i(j) \le k < F_i(j) + a_i^j)$
$(k+1, j+1)$	$\lambda_i(1 - \mathcal{P}_{i,j})$	$(1 \le j < K_i) \wedge (F_i(j) \le k < F_i(j) + a_i^j)$
$(k-1, j)$	$j\mu_i$	$(k \ge 1) \wedge (1 < j \le K_i) \wedge (k - 1 > R_i(j-1))$
$(k-1, j-1)$	$j\mu_i$	$(k \ge 1) \wedge (1 < j \le K_i) \wedge (k - 1 = R_i(j-1))$
$(k-1, j)$	μ_i	$(j = 1) \wedge (k \ge 1)$

(5)

We now proceed to a more detailed description of our iterative solution technique for the *multi-class* system. We do this under the assumption that, given $\boldsymbol{P}_i$, we know how to construct $\mathcal{M}_i$ (using Equation (5) above) and compute $\bar{\pi}_i$ (the steady state probability vector corresponding to $\mathcal{M}_i$). The procedure for computing $\bar{\pi}_i$, is given[6] in Section 3.5.

3.3 Iterative Computation

First, note that in general, there are two cases to consider here:

Case 1: $\sum_{i=1}^{N} K_i \le K$; that is, we have a "trivial" case, where the classes do not interfere with each other, and we can solve each individual class model once (i.e., no need for iteration) using the procedure given in Section 3.5 with $\mathcal{P}_{i,j} = 0$, $\forall i, j$.

Case 2: $\sum_{i=1}^{N} K_i > K$, where it is possible that an attempt at server allocation for class i may fail because all K servers in the system are currently allocated. As described above, in this case a form of blocking occurs and we solve the model using our iterative approach outlined in Section 3.1 whose details are now presented below.

[5]Note that, the transition rates described here are a function of the blocking probabilities, $\mathcal{P}_{i,l}$, which change from iteration to iteration, as outlined above; however, for simplicity of notation, we do not indicate the iteration step number in the description of the transition structure of a class i model.

[6]The motivation for first discussing the iterative technique is to simplify the presentation of our approach.

Note also that, the main difficulty in the iterative technique outlined in Section 3.1 is in determining an appropriate procedure for computing the blocking probabilities which capture the class interaction, i.e., the probabilities that, upon a forward threshold crossing, it is not possible to allocate another server to class i. Recall that, during the n^{th} iteration ($n \ge 0$), $\mathcal{P}_{i,l}^{(n)}$ is the blocking probability of class i ($1 \le i \le N$) to which l servers have already been allocated ($1 \le l \le K_i - 1$). Before we proceed, let us state the following definitions.

DEFINITION 1. *Let $\mathcal{X}$ and $\mathcal{Y}$ be two non-negative random variables having values in $\{1, 2, ...\}$ and let $\pi_{\mathcal{X}}$ and $\pi_{\mathcal{Y}}$ be their respective probability mass functions. Let $\mathcal{Z}$ be another non-negative random variable where $\mathcal{Z} = \mathcal{X} + \mathcal{Y}$; then $\pi_{\mathcal{Z}} = \pi_{\mathcal{X}} \otimes \pi_{\mathcal{Y}}$ where $\otimes$ is the convolution operator.*

DEFINITION 2. *Let $\mathcal{X}$ be a non-negative random variable having values in $\{1, 2, \ldots, \}$ and let $\pi_{\mathcal{X}}$ be its probability mass function. Let*

$$\mathcal{X}' = \begin{cases} \mathcal{X} & \text{if } L_1 \le \mathcal{X} \le L_2 \\ 0 & \text{otherwise.} \end{cases}$$

Then the probability mass function of $\mathcal{X}'$, denoted by $\pi_{\mathcal{X}'}$, is equal to $g(\pi_{\mathcal{X}}, L_1, L_2)$ where function g is defined such that:

$$\pi_{\mathcal{X}'}[k] = \begin{cases} \frac{\pi_{\mathcal{X}}[k]}{\sum_{m=L_1}^{L_2} \pi_{\mathcal{X}}[m]} & \text{if } L_1 \le k \le L_2 \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

Let $\bar{\pi}_i^{(n)}[k, j]$ be the steady state probability of class i having k customers ($k \ge 0$) in the system and an allocation of j servers ($1 \le j \le K_i$), computed during the n^{th} iteration. Let $\pi_i^{(n)}$ denote the steady state probability vector of the number of servers allocated to class i, where $\pi_i^{(n)}[j]$ denotes the steady state probability of j servers having been allocated to class i, as computed during the n^{th} iteration. Thus, we have:

$$\pi_i^{(n)}[j] = \sum_k \bar{\pi}_i^{(n)}[k, j] \quad (7)$$

Finally, let $\boldsymbol{Q}_i^{(n)}$ be the transition rate matrix corresponding to the class i model $\mathcal{M}_i^{(n)}$, during the n^{th} iteration, which is computed using the transition structure of $\mathcal{M}_i^{(n)}$ given in Equation (5) and $\mathcal{P}_{i,l}^{(n-1)}$, where $1 \le l \le K_i - 1$. Then, the iterative procedure is as follow:

1. *Initialization step:* set $n = 0$ and set $\mathcal{P}_{i,l}^{(0)} = 0$ for $1 \leq l < K_i$. Given these initial values of blocking probabilities, for each class i, we can construct $\boldsymbol{Q}_i^{(0)}$ using the transition structure given in Equation (5) and then compute $\tilde{\pi}_i^{(0)}$ using the procedure given in Section 3.5. Once we compute the steady state probability vector $\tilde{\pi}_i^{(0)}$ for each class i, we can then compute their respective server allocation probability vectors, $\pi_i^{(0)}$'s, using Equation (7). The $\pi_i^{(0)}$'s are in turn needed in the computation of the blocking probabilities, $\mathcal{P}_{i,l}^{(1)}$'s (step 2 below).

2. *Updating of blocking probabilities step:* $n = n + 1$, and

$$\mathcal{P}_{i,l}^{(n)} = \begin{cases} 0 & \text{if } K \geq \sum_{j=1}^{N} K_j \\ 0 & \text{if } K - l > \sum_{j=1, j\neq i}^{N} K_j \\ \Gamma(i,l,n) & \text{otherwise} \end{cases} \quad (8)$$

The first condition in Equation (8) indicates that the system has a sufficient number of servers for all classes (we include this for completeness). The second condition indicates that the system has sufficient resources to allocate at least one more server to class i without affecting the maximum possible server allocation of other classes. In the last condition, the Γ function is used to compute the blocking probability, at iteration n, for class i which has l servers already allocated to it.

$\Gamma(i,l,n)$ can be computed as follows. Let $\mathcal{A}_m(i,l,n)$ be the random variable, at iteration n, denoting server allocation of class m, when class i has been allocated l servers. Let $\Upsilon_m(i,l,n)$ be the probability mass function of $\mathcal{A}_m(i,l,n)$. Then we have:

$$\Upsilon_m(i,l,n) = g(\pi_m^{(n-1)}, 1, L_m) \quad (9)$$

for $m = \{1, 2, \ldots, i-1, i+1, \ldots, N\}$ where function g is defined through Equation (6) and L_m is as follows:

$$L_m = \begin{cases} K_m & \text{if } K-l-(N-2) \geq K_m \\ K-l-(N-2) & \text{otherwise} \end{cases} \quad (10)$$

and $\pi_m^{(n-1)}$ in Equation (9) is computed using Equation (7). The normalization in Equation (9) is used to account for the fact that if we know that the system already allocated l servers to class i, then the system only has $(K-l)$ servers remaining. Out of these $(K-l)$ remaining servers, the system needs to allocate $(N-2)$ to customers that are neither in class i nor in class m (i.e., the system allocates at least one server to each class). Therefore, if the system potentially has at least K_m available servers, then $\mathcal{A}_m(i,l,n)$ can have values in $\{1, \ldots, K_m\}$; otherwise, the random variable $\mathcal{A}_m(i,l,n)$ can only take on values in $\{1, 2, \ldots, K-l-(N-2)\}$.

Let $\mathcal{B}(i,l,n)$ be a non-negative random variable, at iteration n, denoting the server allocation of all classes *except* class i, where class i already has l servers allocated to it. Let $\Psi(i,l,n)$ be the probability mass function of $\mathcal{B}(i,l,n)$. Then we have:

$$\Psi(i,l,n) = g\Big(\big(\Upsilon_1(i,l,n) \otimes \Upsilon_2(i,l,n) \cdots \Upsilon_{i-1}(i,l,n) \otimes \Upsilon_{i+1}(i,l,n) \otimes \cdots \otimes \Upsilon_N(i,l,n)\big), N-1, K-l\Big) \quad (11)$$

The normalization in Equation (11) is used to account for the fact that if the system has already allocated l servers to class i, then the number of servers that have been allocated to other classes can only range in $\{N-1, N, \ldots, K-l\}$.

Lastly, $\Gamma(i,l,n)$, the function used to compute blocking probabilities, at iteration n, corresponding to class i with l allocated servers is:

$$\Gamma(i,l,n) = \Big(\Psi(i,l,n; K-l)\Big)\Big(\pi_i^{(n-1)}[l]\Big) \quad (12)$$

where $\Psi(i,l,n;K-l) = \text{Prob}[\mathcal{B}(i,l,n) = K-l]$ and $\Psi(i,l,n)$ is computed using Equation (11).

3. *Updating of individual class models step:* given the blocking probabilities $\mathcal{P}_{i,l}^{(n)}$ of class i in Equation (8), we can compute the new rate matrix $\boldsymbol{Q}_i^{(n)}$ (based on the transition structure given in Equation (5)) and then compute the corresponding steady state probabilities $\tilde{\pi}_i^{(n)}$ (using the procedure given in Section 3.5) as well as $\pi_i^{(n)}$, the probability vector of server allocation of class i (using Equation (7)). (The $\pi_i^{(n)}$'s will in turn be needed in the updating of the blocking probabilities, $\mathcal{P}_{i,l}^{(n+1)}$'s (step 2 above).)

4. *Test of convergence step:* if $|\mathcal{P}_{i,l}^{(n)} - \mathcal{P}_{i,l}^{(n-1)}| \leq \epsilon$ for each class i, $1 \leq i \leq N$, and each l, $1 \leq l \leq K_i - 1$, then stop. Otherwise, go to step 2 and continue iterating.

3.4 Computation of Performance Measures

In this section we briefly discuss computation of performance measures. Given the steady state probabilities $\tilde{\pi}_i$, $i = 1, \ldots, N$, computed using the iterative approach described above, we can compute various performance measures of interest. More specifically, for each class i we can compute performance measures which can be expressed in the form of a Markov reward function, $\mathcal{R}_i$, where

$$\mathcal{R}_i = \sum_{k,j} \tilde{\pi}_i[k,j] R_i(k,j)$$

and $R_i(k,j)$ is the reward for state (k,j) of class i. Some useful performance measures include: (a) expected number of customers of class i, (b) expected response time for customers of class i, (c) probability of dropping a customer of class i upon its arrival, (d) throughput of class i customers, and so on.

For instance, let $E[N_i]$ and $E[T_i]$ denote the expected number of customers and the expected response time, respectively, of the class i model, corresponding to the Markov process $\mathcal{M}_i$. Then $E[N_i]$ can be expressed as $\sum_{k,j} k\tilde{\pi}_i[k,j]$. (A more detailed expression for $[N_i]$ is given in [4].) Of course, using Little's result [15], we have $E[T_i] = \frac{1}{\lambda_i^*} E[N_i]$, where λ_i^* is the class i throughput. To compute λ_i^* we need to account for the customers that are dropped from the system (see Section 2). Hence, $\lambda_i^* = \lambda_i(1 - \sum_{j=1}^{K_i-1} \mathcal{P}_{i,j}\tilde{\pi}_i[F_i(j) + a_i^j, j])$.

We believe that the more interesting performance measures are those computed on a per class basis, since a useful part of studying performance of multi-class threshold-based systems is to discover the effect that the various classes have on

one another. Hence above (and in Section 4) we have concentrate on per class performance measures. However, we can also use these to compute overall system performance measures, for instance, as a weighted average of the individual class performance measures. For example, we can compute the expected system response time, $E[T]$, as follows:

$$E[T] = \frac{\lambda_1^*}{\lambda^*}E[T_1] + \frac{\lambda_2^*}{\lambda^*}E[T_2] + \cdots + \frac{\lambda_N^*}{\lambda^*}E[T_N]$$

where $\lambda^* = \sum_{i=1}^{N} \lambda_i^*$.

3.5 Analysis of the Individual Class Model

In this section we briefly summarize the solution technique for the individual class model which was defined in Section 3.2. Specifically, we use the single class solution technique we derived in [16] with some modifications needed to account for the structure of the multi-class model. Since these modification are mostly straightforward, we only summarize the solution technique in this section, and give the details in [4].

The general approach is as follows. As already stated, we model the class i queueing system as a Markov process, $\mathcal{M}_i$, where: (1) the main goal is to compute the steady state probabilities of the Markov process and use these to compute various performance metrics of interest and (2) the main difficulty is that the Markov process is infinite (see Section 3.2) and thus "difficult" to solve using a "direct" approach[7].

As is often done in these cases, we need to look for special structure that might exist in the Markov process; specifically, we take advantage of the stochastic complementation technique [17]. The basic approach to computing the steady state probabilities of the Markov process and the corresponding performance measures is as follows. We first partition the state space of the original Markov process $\mathcal{M}_i$ into disjoint sets. Using the concept of stochastic complementation, for each set, we compute the conditional steady state probability vector, given that the original Markov process $\mathcal{M}_i$ is in that set. (A relatively simple construction of the stochastic complement is possible due to the special structure that exists in the individual class models; specifically we exploit the "single entry" structure as in [16].) By applying the state aggregation technique [1], we aggregate each set into a single state and then compute the steady state probabilities for the aggregated process, i.e., the probabilities of the system being in any given set. Lastly, we apply the disaggregation technique [1] to compute the individual (unconditional) steady state probabilities of the original Markov process $\mathcal{M}_i$. These can in turn be used to compute various performance measures of interest. (Refer to [4] for a detailed derivation of the solution of $\mathcal{M}_i$.)

4. NUMERICAL EXAMPLES AND VALIDATION OF APPROXIMATION

In this section, we present numerical examples which illustrate (1) the accuracy of our iterative solution technique as compared with simulation as well as (2) the use of our solution technique in studying performance of designs of threshold-based systems with hysteresis behavior.

[7]We could consider finite versions of the model or truncation of the infinite version [11]; however, in either case the Markov process would still be very large and the computational complexity of a "direct" solution for a reasonable size system still high.

Accuracy of our approach.

We begin with the illustration of accuracy of our iterative solution. Thus, in addition to solving each example model, represented by the Markovian process $\mathcal{M}$, using our iterative approach (as described in Section 3) we also simulate $\mathcal{M}$, for the purpose of validating this solution technique. In all experiments presented here, our iterative approach uses $\epsilon = 0.0000001$ (refer to Section 3 for details). Note that, in this section, we use the mean response time of each class i, as the performance metric of interest. Lastly, parameter settings[8] for all test cases presented in this section are listed in Table 1.

Figures 2 and 3 depict the difference in results obtained through simulation and through the iterative approach.

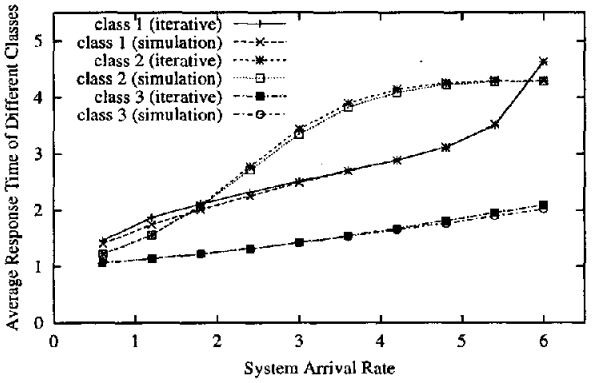

Figure 2: Test Case #1A.

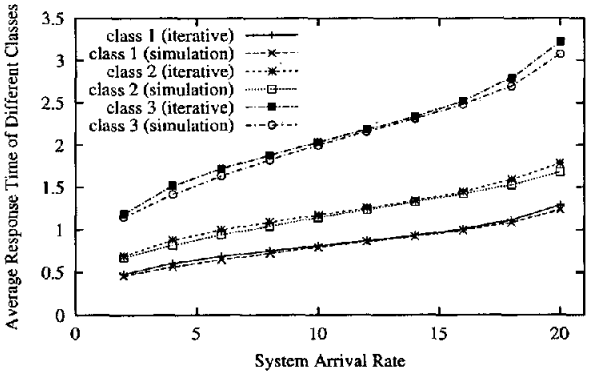

Figure 3: Test Case #1B.

As can be seen from these figures, the difference between the two results is small (e.g., in the case of Figure 2, the largest difference is $\approx$ 5%). Given such small differences, which are difficult to assess using graphs, we present the remainder of the accuracy related experiments using tables.

[8]For ease of specification, we use the following notation $a_i = [a_i^1, \ldots, a_i^{K_i-1}]$ to indicate the a_i^j values for each class i with j allocated servers (see Section 2).

Test Cases	Parameters Settings
Test Case #1A	$K = 10, K_1 = K_2 = K_3 = 4, \alpha_1 = 0.6, \alpha_2 = 0.3, \alpha_3 = 0.1, \mu_1 = \mu_2 = \mu_3 = 1.0$ $\boldsymbol{F}_1 = [4, 8, 12], \boldsymbol{R}_1 = [2, 6, 10], \boldsymbol{F}_2 = [8, 12, 16], \boldsymbol{R}_2 = [5, 9, 13],$ $\boldsymbol{F}_3 = [6, 10, 14], \boldsymbol{R}_3 = [3, 7, 11], a_1 = [2, 2, 2], a_2 = [2, 2, 2], a_3 = [2, 2, 2].$
Test Case #1B	$K = 10, K_1 = K_2 = K_3 = 4, \mu_1 = 3.0, \mu_2 = 2.0, \mu_3 = 1.2$ $\boldsymbol{F}_1 = \boldsymbol{F}_2 = \boldsymbol{F}_3 = [4, 8, 12], \boldsymbol{R}_1 = \boldsymbol{R}_2 = \boldsymbol{R}_3 = [2, 6, 10]$ $a_1 = a_2 = a_3 = [2, 2, 2], \alpha_1 = 0.5, \alpha_2 = 0.3, \alpha_3 = 0.2$
Test Case #2	$K = 10, K_1 = 4, K_2 = 4, K_3 = 4, \mu_1 = \mu_2 = \mu_3 = 1.0,$ $\boldsymbol{F}_1 = \boldsymbol{F}_2 = \boldsymbol{F}_3 = [6, 10, 15], \boldsymbol{R}_1 = \boldsymbol{R}_2 = \boldsymbol{R}_3 = [4, 5, 8], a_1 = a_2 = a_3 = [3, 3, 3].$
Test Case #3	$K = 10, K_1 = 3, K_2 = 3, K_3 = 8, \mu_1 = \mu_2 = \mu_3 = 1.0,$ $\boldsymbol{F}_1 = [6, 10], \boldsymbol{R}_1 = [4, 7], a_1 = [3, 3], \boldsymbol{F}_2 = [6, 10], \boldsymbol{R}_2 = [4, 7], a_2 = [3, 3],$ $\boldsymbol{F}_3 = [6, 10, 14, 18, 22, 26, 30], \boldsymbol{R}_3 = [4, 7, 10, 13, 16, 19, 21], a_3 = [3, 3, 3, 3, 3, 3, 3].$
Test Case #4	$K = 12, K_1 = 3, K_2 = 3, K_3 = 3, K_4 = 5, a_1 = [3, 3], a_2 = [3, 3], a_3 = [3, 3],$ $a_4 = [3, 3, 3, 3], \mu_1 = \mu_2 = \mu_3 = 1.0, \boldsymbol{F}_1 = [6, 10], \boldsymbol{R}_1 = [4, 7], \boldsymbol{F}_2 = [4, 8],$ $\boldsymbol{R}_2 = [2, 4], \boldsymbol{F}_3 = [8, 12], \boldsymbol{R}_3 = [6, 9], \boldsymbol{F}_4 = [5, 9, 13, 17], R_4 = [3, 6, 9, 12].$
Test Case #5	$K = 12, K_1 = 3, K_2 = 3, K_3 = 3, K_4 = 6, \mu_1 = \mu_2 = \mu_3 = 1.0$ $\boldsymbol{F}_1 = [6, 10], \boldsymbol{R}_1 = [4, 7], a_1 = [3, 3], \boldsymbol{F}_2 = [4, 8], \boldsymbol{R}_2 = [2, 4], a_2 = [3, 3],$ $\boldsymbol{F}_3 = [8, 12], \boldsymbol{R}_3 = [6, 9], a_3 = [3, 3],$ $\boldsymbol{F}_4 = [5, 9, 13, 17, 21], \boldsymbol{R}_4 = [3, 6, 9, 12, 15], a_4 = [3, 3, 3, 3, 3].$
Test Case #6	$K = 10, K_1 = K_2 = K_3 = 4, \mu_1 = \mu_2 = \mu_3 = 1.0,$ $\boldsymbol{F}_1 = \boldsymbol{F}_2 = \boldsymbol{F}_3 = [6, 10, 15], \boldsymbol{R}_1 = \boldsymbol{R}_2 = \boldsymbol{R}_3 = [4, 5, 8],$ $a_1 = a_2 = a_3 = [3, 3, 3], \lambda_1 = \lambda_2 = 2.0.$
Test Case #7	$K = 8, K_1 = K_2 = K_3 = K_4 = 3, \mu_1 = \mu_2 = \mu_3 = 1.0$ $\boldsymbol{F}_1 = \boldsymbol{F}_2 = \boldsymbol{F}_3 = [10, 20],$ $\boldsymbol{R}_1 = \boldsymbol{R}_2 = \boldsymbol{R}_3 = [5, 15], a_1 = a_2 = a_3 = a_4 = [2, 2], \alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = 0.25.$ Class 4 has three configurations: (A) $\boldsymbol{F}_4 = [5, 10], \boldsymbol{R}_4 = [3, 7];$ (B) $\boldsymbol{F}_4 = [7, 14], \boldsymbol{R}_4 = [4, 11];$ (C) $\boldsymbol{F}_4 = [10, 20], \boldsymbol{R}_4 = [5, 15].$
Test Case #8	$K = 10, K_1 = 6, K_2 = 4, K_3 = 2, \alpha_1 = 0.6, \alpha_2 = 0.3, \alpha_3 = 0.1,$ $a_1 = [2, 2, 2, 2, 2], a_2 = [2, 2, 2], a_3 = [2], \mu_1 = \mu_2 = \mu_3 = 1.0.$ There are two configurations: (A) $\boldsymbol{F}_1 = [4, 8, 12, 16, 20], \boldsymbol{R}_1 = [2, 6, 10, 14, 18],$ $\boldsymbol{F}_2 = [6, 10, 14], \boldsymbol{R}_2 = [4, 8, 12], \boldsymbol{F}_3 = [8], \boldsymbol{R}_3 = [6];$ (B) $\boldsymbol{F}_1 = [4, 8, 12, 16, 20], \boldsymbol{R}_1 = [2, 6, 10, 14, 18],$ $\boldsymbol{F}_2 = [8, 12, 16], \boldsymbol{R}_2 = [6, 10, 14], \boldsymbol{F}_3 = [8], \boldsymbol{R}_3 = [6].$
Test Case #9	$K = 10, K_1 = K_2 = K_3 = 4, a_1 = a_2 = a_3 = [2, 2, 2],$ $\mu_1 = 1.0, \mu_2 = 10.0, \mu_3 = 100.0,$ $\boldsymbol{F}_1 = \boldsymbol{F}_2 = \boldsymbol{F}_3 = [4, 8, 12], \boldsymbol{R}_1 = \boldsymbol{R}_2 = \boldsymbol{R}_3 = [2, 6, 10].$

Table 1: Data Sets.

Tables 2-6 illustrate several other experiments of validating the accuracy of our technique. (Due to the large size of the tables we only give the iterative result and the percentage error.) In all cases, the percentage error (%E) is defined as:

$$\%E = \frac{|\text{simulation result} - \text{iterative result}|}{\text{simulation result}} \times 100\% \quad (13)$$

Although Table 2 is not the most interesting case from a design point of view, it is used to illustrate that our iterative approach "does the right thing", i.e., it produces the same results for all classes, for a system where all classes behave identically. Furthermore, as can be seen from Tables 2-6 the accuracy of our technique is good, even under high contention.

Note that, we have performed many more experiments than we have been able to include in the paper. The results of those experiments are similar to the ones included here and can be found in a technical report [4]. Overall, the percentage error in most cases we tested was within 5%, with few cases having an error of greater than 12%.

As is probably expected, in our experiments, the higher error cases corresponded to fairly high contention cases. These are also the cases that likely corresponded to "poor" designs where a reduction in contention for resources between classes is needed in order to obtain a system with good performance characteristics. In most of our experiments (some of which are presented below), the performance improvements that could be obtained, for instance, through better threshold settings, were significantly higher (percentage-wise) than the loss in accuracy due to our approximation. Hence, this is a good indication that our iterative technique is a useful tool for fast and fairly accurate assessment of threshold-based designs that can be used, for instance, for searching for good threshold settings. Next, we illustrate some of the performance tradeoffs and designs that can be studied using our technique.

Response time behavior.
We begin by illustrating, in Figure 4, the somewhat "peculiar" response time behavior of threshold-based resource management techniques as well as their potential utility in dynamic resource management of systems. In this figure we depict a three class system where we fix the arrival rate of classes 1 and 2 (at $\lambda_1 = \lambda_2 = 2.0$) and vary the arrival rate

λ_1	λ_2	λ_3	$E[T_1]$ (iterative)	$E[T_2]$ (iterative)	$E[T_3]$ (iterative)	% error (class 1)	% error (class 2)	% error (class 3)
0.40	0.40	0.40	1.630272	1.630272	1.630272	1.202181	0.735494	1.153579
0.80	0.80	0.80	2.597491	2.597491	2.597491	3.058886	2.665640	3.148594
1.20	1.20	1.20	3.156590	3.156590	3.156590	2.877489	2.619260	2.598514
1.60	1.60	1.60	3.304191	3.304191	3.304191	2.000313	1.749316	1.993638
2.00	2.00	2.00	3.321952	3.321952	3.321952	1.630424	1.625574	1.719706
2.40	2.40	2.40	3.379876	3.379876	3.379876	2.282249	1.994334	2.188239
2.80	2.80	2.80	3.592408	3.592408	3.592408	3.792533	3.334905	3.770227
3.20	3.20	3.20	3.981657	3.981657	3.981657	1.820473	1.245720	1.811621
3.60	3.60	3.60	5.010319	5.010319	5.010319	0.601689	1.198128	1.858318

Table 2: Test Case #2.

λ_1	λ_2	λ_3	$E[T_1]$ (iterative)	$E[T_2]$ (iterative)	$E[T_3]$ (iterative)	% error (class 1)	% error (class 2)	% error (class 3)
0.30	0.30	0.30	1.421688	1.421688	1.421680	0.413537	0.181734	0.318665
0.60	0.60	0.60	2.122039	2.122039	2.122039	2.657185	2.095714	2.378890
0.90	0.90	0.90	2.795585	2.795585	2.795464	3.284193	2.855018	2.994942
1.20	1.20	1.20	3.180244	3.180244	3.178452	2.371027	1.896997	1.832370
1.50	1.50	1.50	3.357821	3.357821	3.344765	1.286876	0.882332	1.135763
1.80	1.80	1.80	3.470854	3.470854	3.412461	0.711188	0.498285	0.647543
2.10	2.10	2.10	3.641093	3.641093	3.441965	0.564150	0.070303	0.557836
2.40	2.40	2.40	4.068188	4.068188	3.462714	0.491789	0.814125	0.468379
2.70	2.70	2.70	5.600409	5.600409	3.512400	0.911634	2.755618	0.956535
2.70	2.70	3.20	5.611609	5.611609	3.646626	0.158437	3.637028	2.843098
2.70	2.70	4.00	5.647604	5.647604	3.935655	0.236337	3.279166	5.373869
2.70	2.70	4.80	5.693371	5.693371	4.113839	0.126605	1.469008	1.825511
2.70	2.70	5.60	5.733468	5.733468	4.178537	0.048777	0.166028	0.909810
2.70	2.70	6.40	5.763689	5.763689	4.135517	0.767475	1.791951	1.555702
2.70	2.70	7.20	5.785786	5.785786	4.301380	2.732223	2.217973	1.388458

Table 3: Test Case #3.

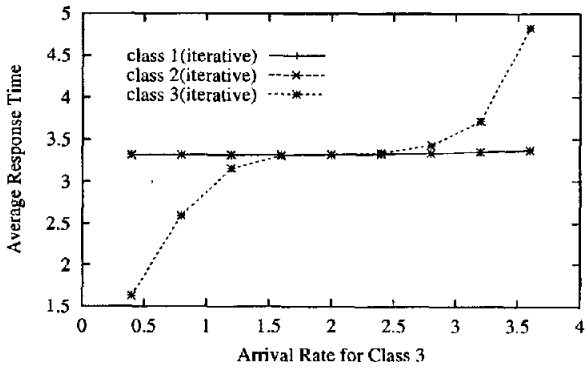

Figure 4: Test Case #6.

of class 3. We can make a couple of observation here: (1) the response time curve of class 3 shows the "peculiar" response time behavior mentioned above, i.e., it first increases, then "flattens" out and then increases again, as a function of increasing arrival rate; (2) the response time curves of classes 1 and 2 are flat, i.e., the increased workload in the system due to class 3 does not appear to affect the response time of classes 1 and 2, even though there is contention for resources between all classes.

The first observation is, of course, due to the fact that in threshold-based systems the response time can improve at higher loads due to the crossing of a forward threshold (which does not occur at lower loads). The second observation suggests that threshold-based techniques can reduce the sensitivity of performance characteristics of a class of customers to the workload of other classes without having to partition resources statically.

Performance sensitivity.

We continue in Figures 5 and 6 with the demonstration of the last point, where we illustrate that it is worth while to study the behavior of threshold-based systems, for instance, to search for better threshold settings, as the changes in performance (due to better parameter settings) are often significant. Given a fast and fairly accurate analytical solution technique (such as ours), these studies can be made efficient.

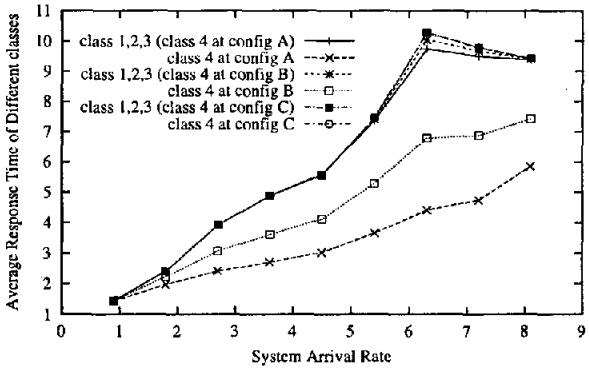

Figure 5: Test Case #7.

In Figure 5 all classes begin with the same characteristics. We then vary the threshold settings of class 4 to experiment with the effect this has on class 4 performance as well as on the performance of the remaining classes. As can be seen from this figure, we are able to improve class 4 performance with "more aggressive" threshold settings, without it having a significant effect (in most cases) on the performance of the remaining classes. It is interesting to note that in places

λ_1	λ_2	λ_3	λ_4	$E[T_1]$ (iterative)	$E[T_2]$ (iterative)	$E[T_3]$ (iterative)	$E[T_4]$ (iterative)	% error (class 1)	% error (class 2)	% error (class 3)	% error (class 4)
0.90	0.90	1.80	4.50	2.797506	1.994472	4.529838	4.374533	2.961873	6.642513	0.469696	0.212680
0.90	1.80	1.80	4.50	2.853014	2.441160	4.543077	4.375314	4.506275	2.851423	0.630932	0.393236
1.80	1.80	1.80	4.50	3.581616	2.504496	4.671432	4.384649	2.517501	3.911467	2.294438	0.122387
2.70	2.70	2.70	4.50	5.734653	4.849304	6.490237	4.530748	0.130256	1.235332	1.043941	2.141307

Table 4: Test Case #4.

λ_1	λ_2	λ_3	λ_4	$E[T_1]$ (iterative)	$E[T_2]$ (iterative)	$E[T_3]$ (iterative)	$E[T_4]$ (iterative)	% error (class 1)	% error (class 2)	% error (class 3)	% error (class 4)
0.90	0.90	1.80	5.40	2.843976	2.017333	4.543616	4.177035	4.322296	7.654710	0.677994	6.927892
0.90	1.80	1.80	5.40	3.108633	2.510323	4.687938	4.186058	11.183943	3.839499	2.575381	2.158903
1.80	1.80	1.80	5.40	3.785969	2.640827	4.934167	4.211717	2.996919	2.662320	3.703350	0.066877
2.70	2.70	2.70	5.40	5.762419	4.868467	6.521253	4.377117	1.494872	0.799943	2.778988	0.505154

Table 5: Test Case #5.

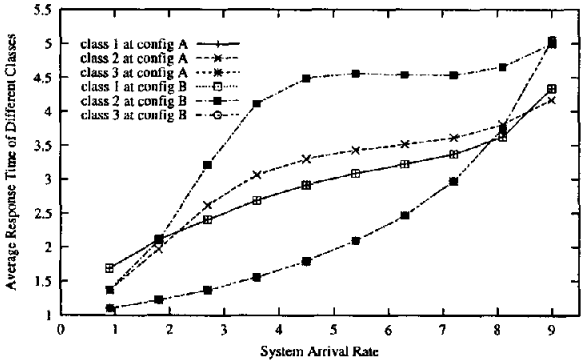

Figure 6: Test Case #8.

where this does have an effect on the performance of the other classes (e.g., at $\lambda = 6.3$) the actual effect is somewhat unexpected. That is, one might expect that the "more aggressive" threshold settings of class 4 might detriment the performance of other classes; however that is not the case here.

In general, it is often difficult to predict the effects of changes in threshold settings, partly due to unusual response time behavior of threshold-based systems and partly due to the interaction of such behavior with systems employing some form of blocking behavior. In the case of Figure 5, this behavior may partly be due to the "more aggressive" threshold settings aiding in processing of the class 4 workload faster and hence resulting in greater resource availability for the other classes as well. Thus, experimentation with effects of threshold settings on system performance is of importance, and the ability to solve the corresponding models efficiently facilitates such experimentation.

In Figure 6 we study the effects of changes in threshold settings of class 2 where all three classes exhibit significantly different characteristics. In this case, partly due to the "more aggressive" settings of thresholds (in configuration A as compared to configuration B), the performance of class 2 improves, but again without having a significant effect on the other two classes.

Lastly, although we have not provided proofs of convergence or of the performance of the iterative technique in terms of convergence rates, we have performed extensive experimentations to gather empirical evidence that our technique does converge, and that it converges fairly quickly. Our experiments indicate that for most test cases the iterative approach converges within approximately 5 to 10 iterations. Based on the "wall clock" time, in most test cases, it produces results more than two orders of magnitude faster than simulation. We note that, in general, theoretical characterization of convergence of iterative techniques which use more than a single parameter (to characterize the intercation between the models), which is the case in our approach, is difficult.

In summary, the main focus of this section was the illustration of utility of our approach in evaluating designs of threshold-based systems, where "good" parameter settings, such as threshold values, constitute a difficult problem. We believe that the efficiency and accuracy of our solution technique facilitates large-scale experimentation with parameter settings and subsequent evaluation of performance of threshold-based designs of systems.

5. CONCLUSIONS

In this paper, we have considered a K-server *multi-class* threshold-based queueing system with hysteresis in which the number of servers, employed for serving customers of each class i, is governed by forward and reverse threshold vectors. The main motivations for using a threshold-based approach was that (a) many applications incur significant server setup, usage, and removal costs and (b) that it is a good approach to dynamically managing a pool of resources between multiple workload classes. The motivation for the use of hysteresis was to control the cost during momentary fluctuations in workload. An important and distinguishing characteristic of our work is that we developed an efficient analytical solution technique for analyzing *multi-class* threshold-based systems with hysteresis behavior, which is needed in modeling of many applications. Specifically, we proposed an iterative solution method, which our empirical evidence indicates to be fast and fairly accurate. Most of our test cases were within 5 percent of the simulation results (used for validation purposes) with more than two orders of magnitude improvement in computation time (as compared to simulation). Furthermore, we studied the performance characteristics of threshold-based systems and showed that proper choices of design parameters, such as threshold values, can produce significant improvements in system per-

λ_1	λ_2	λ_3	$E[T_1]$ (iterative)	$E[T_2]$ (iterative)	$E[T_3]$ (iterative)	% error (class 1)	% error (class 2)	% error (class 3)
3.6	3.6	3.6	4.623164	0.146230	0.010373	2.174990	4.145004	0.192437
3.6	8.0	8.0	4.623164	0.192905	0.010867	0.344933	6.774970	0.110304
3.6	16.0	16.0	4.623164	0.239999	0.011862	2.330279	2.286976	0.550988
3.6	24.0	24.0	4.623164	0.282280	0.012948	3.795961	0.633505	1.696513
3.6	32.0	32.0	4.623163	0.343896	0.014069	0.004456	0.161939	3.182985
3.6	36.0	36.0	4.623165	0.462318	0.014623	0.397470	0.331783	4.019064
3.6	36.0	60.0	4.623203	0.462322	0.017553	1.912505	0.602542	6.828556
3.6	36.0	80.0	4.623463	0.462348	0.019412	1.140015	1.303909	7.189398
3.6	36.0	160.0	4.639706	0.463973	0.027313	1.656744	1.143159	8.942603
3.6	36.0	240.0	4.701579	0.470160	0.034302	8.421698	0.446514	7.297695
3.6	36.0	320.0	4.776830	0.477684	0.038680	2.454293	2.720231	4.943296
3.6	36.0	360.0	4.813562	0.481358	0.048136	6.936106	4.665342	12.454153

Table 6: Test Case #9.

formance. Using this study, we illustrated the utility of our approach in evaluating designs of threshold-based systems, where "good" parameter settings constitute not only an important but a difficult problem. We believe that the efficiency and accuracy of our approach facilitates large-scale experimentation with parameter settings and subsequent performance evaluation studies of threshold-based designs of systems.

Acknowledgements: The authors are grateful to the anonymous referees for their helpful and insightful comments.

6. REFERENCES

[1] P. J. Courtois. *Decomposability : queueing and computer system applications.* ACM monograph series, Academic Press, New York, 1977.

[2] E. de Souza e Silva, S. S. Lavenberg, and R. R. Muntz. A perspective on iterative methods for the approximate analysis of closed queueing networks. In G. Iazeola, P. J. Courtois, and A. Hordijk, editors, *Mathematical Computer Performance and Reliability*, pages 225–244. North Holland, 1984.

[3] L. Golubchik and J. C. Lui. Bounding of performance measures for a threshold-based queueing syst em with hysteresis. In *Proceedings of 1997 ACM SIGMETRICS Conf.*, Seattle, WA, June 1997.

[4] L. Golubchik and J. C. Lui. A fast and accurate iterative solution of a multi-class threshold-based queueing system with hysteresis. Technical Report CS-TR-4115, University of Maryland, March 2000.

[5] S. Graves and J. Keilson. The compensation method applied to a one-product production/inventory problem. *Journal of Math. Operational Research*, 6:246–262, 1981.

[6] O. Ibe. An approximate analysis of a multi-server queueing system with a fixed order of access. Technical Report RC9346, IBM Research, 1982.

[7] O. Ibe and J. Keilson. Multi-server threshold queues with hysteresis. *Performance Evaluation*, 21:185–212, 1995.

[8] O. Ibe and K. Maruyama. An approximation method for a class of queueing systems. *Performance Evaluation*, 5:15–27, 1985.

[9] J. Keilson. *Green's Function Methods in Probability Theory.* Charles Griffin, London, 1965.

[10] J. Keilson. *Markov Chain Models: Rarity and Exponentiality.* Springer, New York, 1979.

[11] F. P. Kelly. *Reversibility and Stochastic Networks.* John Wiley and Sons, 1979.

[12] P. King. *Computer and Communication Systems Performance Modeling.* Prentice-Hall, New York, 1990.

[13] R. Larsen and A. Agrawala. Control of a heterogeneous two-server exponential queueing system. *IEEE Trans. on Software Engineering*, 9:552–526, 1983.

[14] W. Lin and P. Kumar. Optimal control of a queueing system with two heterogeneous servers. *IEEE Trans. on Automatic Control*, 29:696–703, 1984.

[15] J. D. C. Little. A proof of the queueing formula $L = \lambda W$. *Operations Research*, 9:383–387, May 1961.

[16] J. C. Lui and L. Golubchik. Stochastic complement analysis of multi-server threshold queues with hysteresis. *Performance Evaluation*, 35(1-2):19–48, March 1999.

[17] C. Meyer. Stochastic complementation, uncoupling markov chains and the theory of nearly reducible systems. *SIAM Review*, 31(2):240–272, 1989.

[18] D. Mitra and I. Ziedins. Virtual partitioning by dynamic priorities: Fair and efficient resource-sharing by several services. In B. Plattner, editor, *International Zurich Seminar on Digital Communications, Lecture Notes in Computer Science, Broadband Communications*, pages 173–185. Springer, 1996.

[19] D. Mitra and I. Ziedins. Hierarchical virtual partitioning: Algorithms for virtual private networking. In *IEEE GLOBECOM*, pages 1784–1791, 1997.

[20] J. Morrison. Two-server queue with one server idle below a threshold. *Queueing Systems*, 7:325–336, 1990.

[21] R. Nelson and D. Towsley. Approximating the mean time in system in a multiple-server queue that uses threshold scheduling. *Operations Research*, 35:419–427, 1987.