Multi-class Multi-Server Threshold-based Systems: a Study of Non-instantaneous Server Activation*

Cheng-Fu Chou, Leana Golubchik, and John C. S. Lui

Abstract

In this paper, we consider performance evaluation of a system which shares K servers (or resources) among N heterogeneous classes of workloads, where server allocation and de-allocation for class i is dictated by a class specific threshold-based policy with hysteresis control. In particular, the server activation time for class i is *non-instantaneous*. There are many systems and applications where a multi-class threshold-based queueing system can be of great use. One important utility of using threshold-based approaches is in situations where applications may incur server usage costs. In these cases, one needs to consider not only the performance aspects but also the resulting cost/performance ratio. The motivation for using hysteresis control is to reduce the unnecessary cost of server setup (or activation) and server removal (or deactivation) whenever there are momentary fluctuations in workload. Moreover, servers in such systems and applications are often needed by multiple classes of workloads, and hence, it is desirable to find good approaches to sharing server resources among the different classes of workloads, preferably without statically partitioning the server pool among these classes. An important and distinguishing characteristic of our work is that we consider the modeling and analysis of a multi-class system with *non-instantaneous* server activation, which is of use in studying many important applications. The main contributions of this work are (a) in developing an efficient approximation method for solving such models, (b) in verifying the convergence of our iterative method, and (c) in evaluating the resulting accuracy of the technique for computing performance measures of interest, which can subsequently be used in making system design choices.

I. Introduction

In this paper, we consider performance evaluation of a multi-class multi-server system, in which K servers are shared among N heterogeneous classes of workloads and $K \geq N$. In this multi-class multi-server system, servers (resources) are needed by multiple classes of workloads (applications) and we also note that not all classes of requests are operating at high load at the same time. That is, when the traffic loading of class i is low, it is not desirable to operate unnecessarily many servers for that class, due to the incurred usage costs as well as due to the performance consequences of that class under-utilizing the servers while other classes are (possibly) experiencing high traffic workloads. On the other hand,

*C. F. Chou ccf@csie.ntu.edu.tw is with National Taiwan University. and L. Golubchik leana@cs.usc.eduis with University of Southern California and J. C. S. Lui cslui@cse.cuhk.edu.hk is with Chinese University of Hong Kong. Contact author L. Golubchik. Research has been funded in part by the NSF ANI-0070016 grant (part of the joint NSF/CNPq program), by the RGC and the CUHK Mainline Research Grant, as well as by the Integrated Media Systems Center, a National Science Foundation Engineering Research Center, Cooperative Agreement No. EEC-9529152. Any Opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the National Science Foundation.

it is also not desirable for a system to exhibit very long delays, which can result from lack of servers under heavy loads. Therefore, it is an important issue for the system to figure out a good approach to use few resources to serve those requests from all N classes and still attain a good cost/performance ratio instead of statically partitioning the server pool among the classes. To deal with the above issue and efficiently utilize system resources for such multi-class multi-server system, we propose a threshold-based approach to dynamically assign servers to services of different class requests , i.e., how to allocate or de-allocate servers to different classes is governed by a set of thresholds. In this work we use "multi-class multi-server threshold-based system" to refer to our proposed threshold-based system which is able to adaptively share the servers among different workloads without statically partitioning the server pool among the classes.

The motivation for using a threshold-based approach in a system is that applications may incur server usage costs. Thus, one not only needs to consider the performance but also the cost/performance ratio. One approach to improving the cost/performance ratio of a multi-class multi-server system is to *dynamically* react to changes in workload through the use of *thresholds*. For instance, one can maintain the expected response time of an application 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 traffic loading. To possess the above property, 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. Note that in many cases, a "simple" threshold-based system may not suffice since it is prone to workload oscillations. One reason for avoiding oscillations in the above mentioned system is that there may be server setup and removal costs. Such workload oscillations coupled with non-negligible server setup and removal costs can result in a poor cost/performance ratio of a system. Ideally, one wants to add servers only when a system is moving toward a heavily loaded operation region, and one wants to remove servers only when a system is moving to-wards a lightly loaded operation region — it is not appropriate to alter the number of servers during momentary and small changes in workload. Such oscillation behavior can be avoided by adding *hysteresis* behavior. Hence the motivation of this work is looking for efficient analysis techniques of threshold-based queueing systems with hysteresis control.

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 with hysteresis control can be of great use. For example, the Novell file server maintains a memory pool wherein 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 decrease it; the threshold values are based on per-

ceived packet losses due to increases in network traffic activity. Similarly, OS design has been moving toward maintaining a common buffer space pool that can be dynamically managed between the various I/O processes. Another example application is server replication for different classes of Internet services for an overlay network. As the number of requests for a particular class of service increases, the number of servers needed to maintain an acceptable level of quality-of-service guarantees also increases. The use of a threshold-based approach can result in a cost-controlled creation/deletion of servers based on the changes in the workload for a particular class of request. Thus, the model presented in this paper and its efficient solution will be beneficial for many systems and applications.

Now, we begin to give an overview of the multi-class multi-server threshold-based system, which has a total of K servers. In particular, the number of servers employed for servicing class i customers, $i \in \{1, \ldots, N\}$, 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 $R_i(1) < R_i(2) < \cdots < R_i(K_i - 1)$), where K_i is the maximum number of servers that can be allocated to serve class i customers (i.e., the system includes hysteresis control). The service time of class i customers is represented by an exponential random variable with mean μ_i^{-1}. The server activation time for class i is *non-instantaneous*, and it is represented by an exponential random variable with mean β_i^{-1}. In general, $\mu_i \neq \mu_j$ and $\beta_i \neq \beta_j$ for $i, j \in \{1, \ldots, N\}$. Next, we explain how to allocate or de-allocate servers to classes in the system as follows. Initially, each class is allocated a minimum of one server. When a class i customer arrives to an empty system (i.e., when there are no other class i customers), this newly arrived class i request is served by a single server. Arrival of a class i customer when there are already $F_i(j)$ class i customers in the system (with j servers already allocated to serve class i), causes an attempt to allocate one additional server to class i, where $j = 1, \ldots, K_i - 1$. Departure of a class i customer which leaves behind $R_i(j)$ class i customers (with $j + 1$ servers already allocated to this class prior to this departure event), causes a de-allocation of a server from class i, where $j = 1, \ldots, K_i - 1$. In other words, this forces the return of a server, which was earlier allocated to class i, back to the pool of "free" servers which are available for allocation to all classes of customers. Therefore, all N classes of applications share a common pool of K servers, with *dynamic* allocation of servers to classes governed by a set of thresholds with hysteresis behavior. Note that when $\sum_{i=1}^{N} K_i \leq K$, then the classes do not "interfere" with each other since the total peak resource demand is less than or equal to the total number of resources in the system. Of course, a more interesting and challenging case is when $\sum_{i=1}^{N} K_i > K$. In other words, we want to investigate the performance of each class of workload when the number of common servers is less than the total peak resource demand of all classes. By taking advantage of the fact that not all classes are operating at high load at the same time, one may use fewer resources (than with static resource partitioning) to serve requests from all N classes and still achieve a good

cost/performance ratio.

Here arises another challenging problem, i.e., how to determine what are "good" values for these forward and reverse threshold vectors, which are a function of many factors, such as 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. The goal of this work is to develop an efficient method for solution of multi-class multi-server threshold-based queueing system with hysteresis behavior wherein the server activation is *non-instantaneous*. The question of optimal values for the threshold vectors is, in general, a difficult problem and is outside the scope of this paper. On the other hand, we want to point out that efficient model solution techniques can be of great use in evaluating various parameter settings (such as the threshold values). Such analytical models are especially useful at design time, when the speed of evaluation is key. Thus, we believe that our efficient solution method facilitates accessible experimentations for investigating the "quality" of various threshold parameters.

Given the above motivation for the use of threshold-based systems with hysteresis control, we present an efficient technique for solving the corresponding analytical models and computing various performance measures of interest, in the context of *non-instantaneous* server activation. We begin with a very brief survey of some of the existing literature on the topic. A two-server system is considered in [14], [15], [21]. An approximate solution for solving a degenerate form of this problem (where all thresholds are set to zero) is presented in [7], [9]; an approximate solution for a system that employs (non-zero) thresholds is presented in [22] (but without hysteresis). In [8], the authors solve a multi-server threshold-based queueing system with hysteresis, using the Green's function method [6], [10], [11]. In [17] we give a solution of several forms of the single class, multi-server threshold-based queueing system with hysteresis using stochastic complementation [18]. Techniques for computation of bounds for performance measures of single class, multi-server threshold-based queueing systems with hysteresis and non-instantaneous server activation are given in [3]. Lastly, [4] provides a solution technique for multi-class, multi-server threshold-based system with hysteresis and *instantaneous* server activation. In this paper, we extend and generalize that work to *non-instantaneous* server activation; the non-instantaneous server activation can have a significant impact on the system performance (as will be illustrated in Section V) and hence is an important model characteristic to consider. Specifically, in this work we consider and solve a *multi-class,* multi-server threshold-based queueing system with hysteresis control and *non-instantaneous* server activation.

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 this model. Since in many applications, different types of workloads compete for a pool of resources where server activation time is non-negligible (e.g., it takes a non-zero time to replicate and activate a video server in an overlay

network), we consider it an important and distinguishing characteristic of our work. In this paper, 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. We also illustrate the accuracy of our approach, which efficiently computes performance measures of interest, through a set of numerical results. Furthermore, we give an proof and discussion about the convergence of our iterative method to solve the approximate model. This is important since we can get further understanding of our analytic model such that we could construct more precise and efficient analytic model for the multi-class multi-server system. We also believe that the efficiency and accuracy of our iterative approach provides an important step in finding *optimal* threshold values for a multi-class, multi-server threshold based system with hysteresis and *non-instantaneous* server activation. Finally, we note that a variety of iterative approaches have been used in numerous approximation techniques (e.g., refer to [2]). For instance, an iterative technique for a somewhat different control scheme for dynamic resource sharing between multiple classes is employed in [19], [20].

The remainder of this paper is organized as follows. In Section II we give a detailed description of our model. Section III describes our iterative solution approach for this model. The convergence of the iterative method to solve the approximate model is presented in Section IV. The quality of this approach, i.e., its accuracy and utility in system design and evaluation is discussed in Section V through the use of numerical results. Finally, our conclusions are given in Section VI.

II. System Model

The Markovian model for our *multi-class, multi-server threshold-based* queueing system with *hysteresis* control has an infinite state space which can be described as follows. There are K servers in the system with $K \geq N$ where N is the total number of classes of customers. The service time of different classes can be different, and 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, where with probability α_i an arriving customer is of class i and $\sum_{i=1}^{N} \alpha_i = 1$ and $1 \leq i \leq N$. 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 $1 \leq j \leq K_i - 1$. Note that, unlike in [4], [5], the activation of a server for class i is *non-instantaneous* where the server activation time is exponentially distributed with mean β_i^{-1}. As mentioned in Section I, this is motivated by the fact that in many applications addition of a new server takes a non-negligible amount of time.

Each of these K servers 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 and challenging case is where $\sum_{i=1}^{N} K_i > K$. We model this system as a Markovian process $\mathcal{M}$, using two different variations. In the first variation, we constrain the number of class i customers when the number of servers allocated to class i is less than K_i (we motivate this variation below). The Markovian model for this variation is referred to as $\mathcal{M}^a$. In the second variation, we do not use such a constraint, and the Markovian model for this variation is referred to as $\mathcal{M}^b$. We now give a more detailed description of each of these models.

A. $\mathcal{M}^a$ with constraint vector a

The Markovian process $\mathcal{M}^a$, with a constraint vector a, has the following state space $\mathcal{S}^a$:

$$\mathcal{S}^a = \{(n_1, s_1, l_1, \ldots, n_N, s_N, l_N) \mid n_i \geq 0, l_i \in \{1, \ldots, K_i\}, \sum_{i=1}^{N} l_i \leq K, l_i \geq s_i, F_i(l_i) \leq n_i \leq F_i(l_i) + a_i^{l_i},$$
$$s_i \in \{1, 2, \ldots, K_i\}, i = 1, \ldots, N\}$$

where n_i is the number of class i customers in the system, s_i is the number of "busy" (or active) servers currently serving class i customers, and l_i is the number of servers allocated to class i, not all of which may currently be available for service of class i customers since server *activation* process is *non-instantaneous*. 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 $j = l_i$, the system attempts to allocate an additional server for service of class i customers, which is possible *only* when the system has sufficient amount of resources, i.e., if $\sum_{i=1}^{N} l_i < K$. Note that in a system where $\sum_{i=1}^{N} K_i > K$, it may not always be possible to allocate another server since it is possible that 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 $j = l_i$). When $n_i = F_i(j) + a_i^j$, the arriving class i customer is rejected by the system if there is no server available for allocation to class i (i.e., if $\sum l_i = K$). For *correctness*, we assume the following constraint on all a_i^j:

$$F_i(j) + a_i^j < F_i(j+1) + a_i^{j+1} \text{ for } i = 1, 2, \ldots, N \text{ and } j = 1, 2, \ldots, K_i - 1.$$

We also assume that $a_i^{K_i} = \infty$; hence, we have no restrictions on queue length when the maximum number of servers that may be needed by class i have been allocated (i.e., when $l_i = K_i$). The limitation on queue length when $l_i < K_i$ can be motivated by system design considerations. For example, 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, then one may assume that the system is temporarily overloaded and rejection 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 III) 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 $a_i^{K_i} = \infty$).

We now give a detailed description and formal structure for the transitions of $\mathcal{M}^a$. The transitions corresponding to an arrival of a class i customer fall into one of the following categories:

C_1 : no need to allocate another server to class i. The conditions for this category could be (i) the number of class i customers does not cross a corresponding forward threshold or (ii) there are already K_i servers allocated to class i or (iii) there are no available servers in the system due to resource contention among the different classes.

C_2 : a need to allocate another server to class i. The condition for this category is that the number of allocated server for class i is less than K_i and there is an available server in the system and the number of class i customers crosses a forward threshold.

The formal structure for these arrival transitions is as follows.

$$(n_1, s_1, l_1, \ldots, n_i, s_i, l_i, \ldots, n_N, s_N, l_N) \xrightarrow{\lambda\alpha_i} (n_1, s_1, l_1, \ldots, n_i + 1, s_i, l_i, \ldots, n_N, s_N, l_N) \text{ if } C_1 \tag{1}$$

$$(n_1, s_1, l_1, \ldots, n_i, s_i, l_i, \ldots, n_N, s_N, l_N) \xrightarrow{\lambda\alpha_i} (n_1, s_1, \ldots, n_i + 1, s_i, l_i + 1 \ldots, n_N, s_N, l_N) \text{ if } C_2 \tag{2}$$

where conditions C_1 and C_2 are

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

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

The transitions corresponding to a departure of a class i customer fall into one of the following categories:

C_3 : no need to deactivate a server. The conditions for this category are either (i) only one server is allocated to class i or (ii) the number of class i customers does not drop below a reverse threshold.

C_4 : a need to deactivate a server. The condition for this category is that more than one server is allocated to class i and the number of class i customers drops below a corresponding reverse threshold.

The formal structure for these departure transition is as follows.

$$(n_1, s_1, l_1, \ldots, n_i, s_i, l_i, \ldots, n_N, s_N, l_N) \xrightarrow{s_i\mu_i}$$

$$(n_1, s_1, l_1, \ldots, n_i - 1, s_i, l_i, \ldots, n_N, s_N, l_N) \qquad \text{if } C_3 \tag{3}$$

$$(n_1, s_1, l_1, \ldots, n_i, s_i, l_i, \ldots, n_N, s_N, l_N) \xrightarrow{s_i \mu_i}$$

$$(n_1, s_1, l_1, \ldots, n_i - 1, min(s_i, l_i - 1), l_i - 1, \ldots, n_N, s_N, l_N) \qquad \text{if } C_4 \tag{4}$$

where conditions for C_3 and C_4 are

$$\begin{aligned} C_3 &= \Big((n_i > 0) \wedge (l_i = 1)\Big) \vee \Big((n_i > 0) \wedge (n_i - 1 > R_i(l_i - 1)) \wedge (l_i > 1)\Big) \\ C_4 &= \Big((n_i > 0) \wedge (n_i - 1 = R_i(l_i - 1)) \wedge (l_i > 1)\Big). \end{aligned}$$

Lastly, there are transitions corresponding to class i server activations. The condition for these transitions is that the number of active servers is less than the number of allocated servers. The formal structure for these activation transitions is as follows.

$$(n_1, s_1, l_1, \ldots, n_i, s_i, l_i, \ldots, n_N, s_N, l_N) \xrightarrow{(l_i - s_i)\beta_i} (n_1, s_1, l_1, \ldots, n_i, s_i + 1, l_i, \ldots, n_N, s_N, l_N) \qquad \text{if } C_5 \tag{5}$$

where condition $C_5 = (l_i > s_i)$.

B. $\mathcal{M}^b$ without constraint a

The second model variation is $\mathcal{M}^b$, which represents a Markovian process without constraints on the number of class i customers. It has the following state space, $\mathcal{S}^b$:

$$\mathcal{S}^b = \{(n_1, s_1, l_1, \ldots, n_N, s_N, l_N) \,|\, n_i \geq 0, l_i \in \{1, \ldots, K_i\}, l_i \geq s_i, s_i \in \{1, \ldots, K_i\}, \sum s_i \leq K, i = 1, \ldots, N\}$$

where n_i is the number of class i customers in the system, s_i is the number of "busy" (or active) servers currently servicing class i customers, and l_i is the number of servers "expected" to be allocated/activated for class i use — more specifically, according to the threshold vectors, l_i servers *should be* in use by class i customers but may not be, because (i) multiple classes of customers are competing for these servers and (ii) in our model server activation is *non-instantaneous*. Hence, a major difference between $\mathcal{M}^a$ and $\mathcal{M}^b$ is that we use a constraint vector a to limit the queue length when $l_i < K_i$ in $\mathcal{M}^a$ while we do not limit the queue length in $\mathcal{M}^b$. We give a comparison study between these two models in Section V.

We now give a detailed description and formal structure for the transitions of $\mathcal{M}^b$. The transitions corresponding to arrivals of class i customers fall into one of the following categories:
C_1 : no need to increase the number of expected servers (according to the forward threshold vector) for class i. The conditions for this category could be (i) the number of class i customers does not cross a corresponding forward threshold or (ii) the number of expected servers is equal to K_i.

C_2 : a need to increase the number of expected servers for class i. The condition for this category is that the number of expected servers for class i is less than K_i and the number of class i customers crosses a forward threshold.
The formal structure for these arrival transitions is as follows.

$$(n_1, s_1, l_1, \ldots, n_i, s_i, l_i, \ldots, n_N, s_N, l_N) \xrightarrow{\lambda\alpha_i} (n_1, s_1, l_1, \ldots, n_i + 1, s_i, l_i, \ldots, n_N, s_N, l_N) \quad \text{if } C_1 \tag{6}$$

$$(n_1, s_1, l_1, \ldots, n_i, s_i, l_i, \ldots, n_N, s_N, l_N) \xrightarrow{\lambda\alpha_i} (n_1, s_1, \ldots, n_i + 1, s_i, l_i + 1 \ldots, n_N, s_N, l_N) \quad \text{if } C_2 \tag{7}$$

where conditions C_1 and C_2 are

$$C_1 \quad = \quad \Bigg((l_i < K_i) \wedge (n_i < F_i(l_i)) \Bigg) \vee \Bigg(l_i = K_i \Bigg)$$

and

$$C_2 \quad = \quad \Bigg(l_i < K_i \Bigg) \wedge \Bigg(F_i(l_i) = n_i \Bigg).$$

The transitions corresponding departures of class i customers fall into one of the following categories:
C_3 : no need to deactivate a server. The condition for this category is that either (i) the system has only one server for class i or (ii) the number of class i customers does not drop below a reverse threshold.
C_4 : a need to deactivate a server. The condition for this category is that there is more than one server for class i and the number of class i customers drops below a corresponding reverse threshold.
The formal structure for these departure transitions is as follows.

$$\begin{aligned} (n_1, s_1, l_1, \ldots, n_i, s_i, l_i, \ldots, n_N, s_N, l_N) &\xrightarrow{s_i\mu_i} \\ (n_1, s_1, l_1, \ldots, n_i - 1, s_i, l_i, \ldots, n_N, s_N, l_N) \quad &\text{if} \quad C_3 \end{aligned} \tag{8}$$

$$\begin{aligned} (n_1, s_1, l_1, \ldots, n_i, s_i, l_i, \ldots, n_N, s_N, l_N) &\xrightarrow{s_i\mu_i} \\ (n_1, s_1, l_1, \ldots, n_i - 1, min(s_i, l_i - 1), l_i - 1, \ldots, n_N, s_N, l_N) \quad &\text{if} \quad C_4 \end{aligned} \tag{9}$$

where conditions C_3 and C_4 are

$$C_3 \quad = \quad \Bigg((n_i > 0) \wedge (l_i = 1) \Bigg) \vee \Bigg((n_i > 0) \wedge (n_i - 1 > R_i(l_i - 1)) \wedge (l_i > 1) \Bigg)$$

$$C_4 \quad = \quad \Bigg((n_i > 0) \wedge (n_i - 1 = R_i(l_i - 1)) \wedge (l_i > 1) \Bigg).$$

Lastly, there are the transitions corresponding to class i server activation. The condition for these transitions is that (i) the number of active servers is less than the number of "expected" servers and (ii) there

is an available server in the system. The formal structure for these server activation transitions is as follows.

$$(n_1, s_1, l_1, \ldots, n_i, s_i, l_i, \ldots, n_N, s_N, l_N) \xrightarrow{\beta_i} (n_1, s_1, l_1, \ldots, n_i, s_i + 1, l_i, \ldots, n_N, s_N, l_N) \quad \text{if } C_5 \tag{10}$$

where the condition $C_5 = \Big((l_i > s_i) \wedge (\sum_{j=1}^{N} l_j < K)\Big)$.

III. Iterative Method

In this section we describe an *iterative* approach to solving the models presented in Section II. As described in Section II, the corresponding Markov process[1], $\mathcal{M}$, is infinite in multiple dimensions. One can choose to solve this model by (a) simulating the Markovian process $\mathcal{M}$, or (b) looking for special structure, or (c) looking for efficient approximation techniques. Because $\mathcal{M}$ appears to lack sufficient structure for an efficient exact solution technique (e.g., such as the matrix-geometric technique), we describe an approximate iterative solution technique for solving this model. 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) as well as an accurate one (and iterative techniques can often produce fairly accurate results).

A. Basic Approach

Let us first describe the basic approach to solving the above defined Markovian model. We first break up the original model $\mathcal{M}$ into N single class Markovian sub-models, namely, $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_N$ (see Section III-B for a more detailed description of the $\mathcal{M}_i$'s). These N Markovian models are "coupled" via a set of blocking probabilities. 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. Therefore, in general, there is a non-zero probability that class i, which has already (a) allocated s_i servers in the case of $\mathcal{M}^a$ or (b) expected to be allocated/activated s_i servers in the case of $\mathcal{M}^b$, 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. Note that, $s_i = l$ (number of allocated servers for class i) in $\mathcal{M}^a$ while $s_i = l$ (number of expected to be allocated/activated servers for class i) in $\mathcal{M}^b$.

We now formally 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 $\tilde{\boldsymbol{\pi}}_i^{(n)}$. The parameters of each $\mathcal{M}_i^{(n)}$ are computed as a function of blocking probabilities, $\boldsymbol{P}_i^{(n)} =$

[1] In the remainder of the paper, we use $\mathcal{M}$ to represent either $\mathcal{M}^a$ or $\mathcal{M}^b$, for simplicity of exposition.

$\{\mathcal{P}_{i,1}^{(n)}, \mathcal{P}_{i,2}^{(n)}, \ldots, \mathcal{P}_{i,K_i}^{(n)}\}$, which are in turn computed as a function of the steady state probability vector, $\tilde{\boldsymbol{\pi}}_i^{(n-1)}$, obtained during the previous iteration. (We give the details of the construction of $\mathcal{M}_i^{(n)}$ and the computation of $\tilde{\boldsymbol{\pi}}_i^{(n)}$ below[2].) Then, a high level description of our iterative approach is as follows (a more detailed and formal description is given in Section III-C):

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 $\tilde{\boldsymbol{\pi}}_1^{(n)}$, $\tilde{\boldsymbol{\pi}}_2^{(n)}$, ..., $\tilde{\boldsymbol{\pi}}_N^{(n)}$; set $n = n + 1$;
3. Use these steady state probabilities to compute $\boldsymbol{P}_1^{(n)}$, $\boldsymbol{P}_2^{(n)}$, ..., $\boldsymbol{P}_N^{(n)}$;
4. Use these blocking probabilities to update the individual class models, i.e., construct $\mathcal{M}_1^{(n)}$, $\mathcal{M}_2^{(n)}$, ..., $\mathcal{M}_N^{(n)}$, 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.

B. 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. We have K_i servers 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 $\boldsymbol{R}_i = [R_i(1), R_i(2), \ldots, R_i(K_i - 1)]$. where $R_i(j) < F_i(j)$ and $1 \leq j \leq K_i - 1$. And, server activation time is exponentially distributed with rate β_i.

Individual Class Model for $\mathcal{M}^a$

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

$$\mathcal{S}_i = \{(k, j, l) \mid k \geq 0; j, l \in \{1, 2, \ldots, K_i\}, l \geq j,\ F_i(l) \leq k < F_i(l) + a_i^l\}$$

where k is the number of customers in the class i queueing system, j is the number of busy (active) servers, and l is the number of allocated, not all of which may currently be activated due to the *non-instantaneous* nature of server activation in our model. Figure 1 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 in Table I[3], where all transitions are from state (k, j, l), with the state description given above:

[2]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.

[3]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.

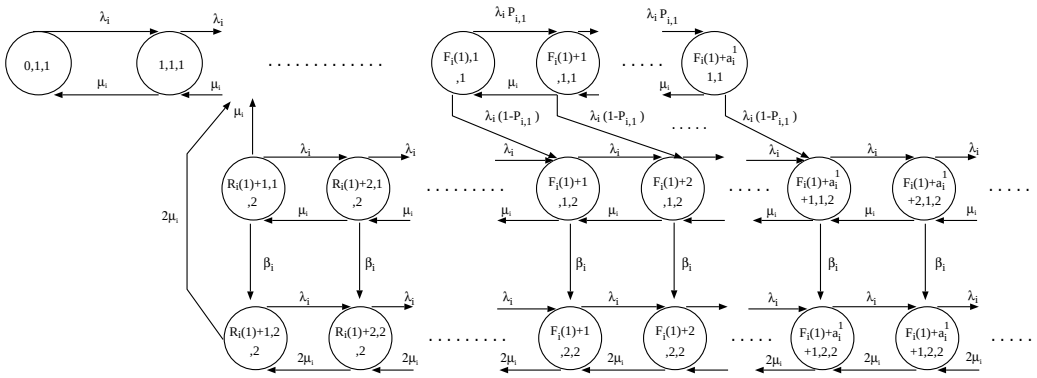


Fig. 1. State transition diagram of $\mathcal{M}^a$ for a class i system with $K_i = 2$.

Next State	Rate	Condition for transition
$(k+1, j, l)$	λ_i	$(1 \leq l < K_i) \wedge (k < F_i(l))$
$(k+1, j, l)$	λ_i	$l = K_i$
$(k+1, j, l)$	$\lambda_i \mathcal{P}_{i,l}$	$(1 \leq l < K_i) \wedge (F_i(l) \leq k < F_i(l) + a_i^l)$
$(k+1, j, l+1)$	$\lambda_i(1 - \mathcal{P}_{i,l})$	$(1 \leq j < K_i) \wedge (F_i(l) \leq k < F_i(l) + a_i^l)$
$(k-1, j, l)$	$j\mu_i$	$(k \geq 1) \wedge (1 < l \leq K_i) \wedge (k-1 > R_i(l-1))$
$(k-1, min(j, l-1), l-1)$	$j\mu_i$	$(k \geq 1) \wedge (1 < l \leq K_i) \wedge (k-1 = R_i(l-1))$
$(k-1, j, l)$	μ_i	$(l = j = 1) \wedge (k \geq 1)$
$(k, j+1, l)$	$(l-j)\beta_i$	$(l > j)$

TABLE I

DESCRIPTION OF STATE TRANSITION FOR $\mathcal{M}^a$

Individual Class Model for $\mathcal{M}^b$

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

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

where k is the number of customers in the class i queueing system, j is the number of busy (active) servers, and l is the number of servers "expected" to be allocated and activated. Figure 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 in Table II, where all transitions are from state (k, j, l), with the state description given above:

Let us now proceed to a more detailed description of the 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 Table I or Table II above) and compute $\tilde{\boldsymbol{\pi}}_i$, the steady state probability vector corresponding to $\mathcal{M}_i$. The procedure

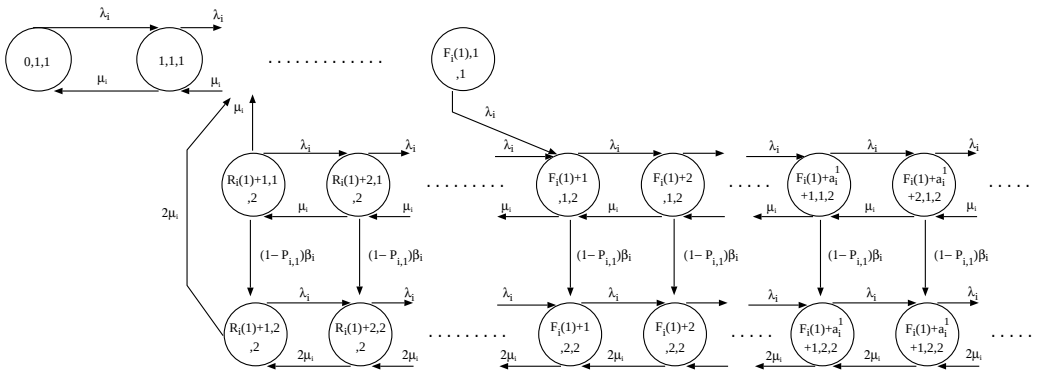


Fig. 2. State transition diagram of $\mathcal{M}^b$ for a class i system with $K_i = 2$.

Next State	Rate	Condition
$(k+1, j, l)$	λ_i	$(1 \leq l < K_i) \wedge (k < F_i(l))$
$(k+1, j, l)$	λ_i	$(l = K_i)$
$(k+1, j, l+1)$	λ_i	$(1 \leq j < K_i) \wedge k = F_i(l)$
$(k-1, j, l)$	$j\mu_i$	$(k \geq 1) \wedge (1 < l \leq K_i) \wedge (k-1 > R_i(l-1))$
$(k-1, min(j, l-1), l-1)$	$j\mu_i$	$(k \geq 1) \wedge (1 < l \leq K_i) \wedge (k-1 = R_i(l-1))$
$(k-1, j, l)$	μ_i	$(l = j = 1) \wedge (k \geq 1)$
$(k, j+1, l)$	$(1 - \mathcal{P}_{i,j})\beta_i$	$(l > j)$

TABLE II

DESCRIPTION OF STATE TRANSITION FOR $\mathcal{M}^b$

for computing $\tilde{\boldsymbol{\pi}}_i$, is given in Section III-E.

C. Iterative Computation

In this subsection, we describe the framework for the iterative procedure. This iterative procedure is similar to our work in [4] but we extend it to handle the case wherein the server activation event is non-instantaneous. First, note that in general, there are two cases to consider here:

case 1: $\sum_{i=1}^{N} K_i \leq K$; 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 III-E with $\mathcal{P}_{i,s_i} = 0, \ \forall i, s_i$.

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 the iterative approach outlined in Section III-A whose details are now presented below.

Note also that the main difficulty in the iterative technique outlined in Section III-A 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 ($n \geq 0$), $\mathcal{P}_{i,s_i}^{(n)}$ is the blocking probability of class i ($1 \leq i \leq N$) to which s_i servers already have been (a) allocated in the case of $\mathcal{M}^a$ or (b) expected to be allocated/activated in the case of $\mathcal{M}^b$. 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 $\boldsymbol{\pi}_{\mathcal{X}}$ and $\boldsymbol{\pi}_{\mathcal{Y}}$ be their respective probability mass functions. Let $\mathcal{Z}$ be another non-negative random variable where $\mathcal{Z} = \mathcal{X} + \mathcal{Y}$; then $\boldsymbol{\pi}_{\mathcal{Z}} = \boldsymbol{\pi}_{\mathcal{X}} \otimes \boldsymbol{\pi}_{\mathcal{Y}}$ 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 $\boldsymbol{\pi}_{\mathcal{X}}$ be its probability mass function. Let

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

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

$$g(\boldsymbol{\pi}_{\mathcal{X}}, L_1, L_2)[k] = \pi_{\mathcal{X}'}[k] = \begin{cases} \frac{\pi_{\mathcal{X}}[k]}{\sum_{m=L_1}^{L_2} \pi_{\mathcal{X}}[m]} & \text{if } L_1 \leq k \leq L_2 \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

Let $\tilde{\pi}_i^{(n)}[k, j, l]$ be the steady state probability of class i having k customers ($k \geq 0$) in the system with j activated servers and l target server allocations (with $1 \leq j \leq l \leq K_i$), computed during the n^{th} iteration. Let $\boldsymbol{\pi}_i^{(n)}$ denote the steady state probability vector of the number of servers allocated to class i, where $\pi_i^{(n)}[l]$ denotes the steady state probability of l servers having been target allocated to class i, as computed during the n^{th} iteration. Thus, we have:

$$\pi_i^{(n)}[l] = \sum_{j=1}^{K_i} \sum_{k} \tilde{\pi}_i^{(n)}[k, j, l] \tag{12}$$

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 (a) Table I in the case of $\mathcal{M}^a$ or (b) Table II in the case of $\mathcal{M}^b$, and $\mathcal{P}_{i,s_i}^{(n-1)}$, where $1 \leq s_i \leq K_i - 1$. Then, the iterative procedure is as follow:

1. *Initialization step:* set $n = 0$ and set $\mathcal{P}_{i,s_i}^{(0)} = 0$ for $1 \leq s_i < K_i$. Given these initial values of blocking probabilities, for each class i, we can construct $\boldsymbol{Q}_i^{(0)}$ using the transition structure given in (a) Table I in the case of $\mathcal{M}^a$ or (b) Table II in the case of $\mathcal{M}^b$, and then compute $\tilde{\boldsymbol{\pi}}_i^{(0)}$ using the procedure given in

Section III-E. Once we compute the steady state probability vector $\tilde{\boldsymbol{\pi}}_i^{(0)}$ for each class i, we can then compute their respective server allocation probability vectors, $\boldsymbol{\pi}_i^{(0)}$s, using Equation (12). The $\boldsymbol{\pi}_i^{(0)}$s are in turn needed in the computation of the blocking probabilities, $\mathcal{P}_{i,s_i}^{(1)}$s (step 2 below).

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

$$\mathcal{P}_{i,s_i}^{(n)} = \begin{cases} 0 & \text{if } K \geq \sum_{j=1}^{N} K_j \\ 0 & \text{if } K - s_i > \sum_{j=1, j\neq i}^{N} K_j \\ \Gamma(i, s_i, n) & \text{otherwise} \end{cases} \tag{13}$$

The first condition in Equation (13) 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 s_i servers already (a) allocated to it in the case of $\mathcal{M}^a$ or (b) expected to be allocated/activated in the cased of $\mathcal{M}^b$.

$\Gamma(i, s_i, n)$ can be computed as follows. Let $\mathcal{A}_m(i, s_i, n)$ be the random variable, at iteration n, denoting server allocation of class m, when class i has been (a) allocated s_i servers in the case of $\mathcal{M}^a$ or (b) expected to be allocated/activated s_i servers in the case of $\mathcal{M}^b$. Let $\Upsilon_m(i, s_i, n)$ be the probability mass function of $\mathcal{A}_m(i, s_i, n)$. Then we have:

$$\Upsilon_m(i, s_i, n) = g(\boldsymbol{\pi}_m^{(n-1)}, 1, L_m) \tag{14}$$

for $m = \{1, 2, \ldots, i-1, i+1, \ldots, N\}$ where function g is defined through Equation (11) and L_m is as follows:

$$L_m = \begin{cases} K_m & \text{if } K - s_i - (N-2) \geq K_m \\ K - s_i - (N-2) & \text{otherwise} \end{cases} \tag{15}$$

and $\boldsymbol{\pi}_m^{(n-1)}$ in Equation (14) is computed using Equation (12). The normalization in Equation (14) is used to account for the fact that if we know that the system already (a) allocated s_i servers to class i in the case of $\mathcal{M}^a$ or (b) expected to be allocated/activated s_i servers to class i in the case of $\mathcal{M}^b$, then the system only has $(K - s_i)$ servers remaining. Out of these $(K - s_i)$ 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 $\mathcal{A}_m(i, s_i, n)$ can have values in $\{1, \ldots, K_m\}$; otherwise, the random variable $\mathcal{A}_m(i, s_i, n)$ can only take on values in $\{1, 2, \ldots, K - s_i - (N-2)\}$. Let $\mathcal{B}(i, s_i, n)$ be a non-negative random variable, at

iteration n, denoting the server allocation of all classes *except* class i, where class i already has s_i servers (a) allocated to it in the case of $\mathcal{M}^a$ or (b) expected to be allocated/activated in the case of $\mathcal{M}^b$. Let $\Psi(i, s_i, n)$ be the probability mass function of $\mathcal{B}(i, s_i, n)$. Then we have:

$$\Psi(i, s_i, n) = g\left((\Upsilon_1(i, s_i, n) \otimes \cdots \otimes \Upsilon_{i+1}(i, s_i, n) \ \ \cdots \otimes \Upsilon_N(i, s_i, n)), N-1, K-s_i\right) \tag{16}$$

The normalization in Equation (16) is used to account for the fact that if the system has already (a) allocated s_i servers to class i in the case of $\mathcal{M}^a$, or (b) expected to be allocated/activated s_i servers to class i in the case of $\mathcal{M}^b$, then the number of servers that have been allocated to other classes can only range in $\{N-1, N, \ldots, K-s_i\}$.

Lastly, $\Gamma(i, s_i, n)$, the function used to compute blocking probabilities, at iteration n, corresponding to class i with (a) s_i allocated servers in the case of $\mathcal{M}^a$ or (b) s_i expected to be allocated/activated servers in the case of $\mathcal{M}^b$ is:

$$\Gamma(i, s_i, n) \ = \ \Psi(i, s_i, n; K - s_i) \tag{17}$$

where $\Psi(i, s_i, n; K - s_i) = \text{Prob}[\mathcal{B}(i, s_i, n) = K - s_i]$ and $\Psi(i, s_i, n)$ is computed using Equation (16).

3. *Updating of individual class models step:* given the blocking probabilities $\mathcal{P}_{i,s_i}^{(n)}$ of class i in Equation (13), we can compute the new rate matrix $\boldsymbol{Q}_i^{(n)}$ (based on the transition structure given in (a) Table I in the case of $\mathcal{M}^a$ or (b) Table II in the case of $\mathcal{M}^b$) and then compute the corresponding steady state probabilities $\tilde{\boldsymbol{\pi}}_i^{(n)}$ (using the procedure given in Section III-E) as well as $\boldsymbol{\pi}_i^{(n)}$, the probability vector of server allocation of class i (using Equation (12)). (The $\boldsymbol{\pi}_i^{(n)}$'s will in turn be needed in the updating of the blocking probabilities, $\mathcal{P}_{i,s_i}^{(n+1)}$'s (step 2 above).)

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

D. Computation of Performance Measures

In this section we briefly discuss computation of performance measures. Due to lack of space, we only present the derivation for model $\mathcal{M}^a$ and we could use the same approach for the derivation for model $\mathcal{M}^b$. Given the steady state probabilities $\tilde{\boldsymbol{\pi}}_i, i = 1, \ldots, N$, 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

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

and $R_i(k, j, l)$ is the reward for state (k, j, l) 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 $\sum_{k,j,l} k\tilde{\pi}_i[k,j,l]$. (A more detailed expression for $E[N_i]$ is given in equation (33) in Appendix A.) Of course, using Little's result [16], 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 II). Hence, $\lambda_i^* = \lambda_i(1 - \sum_{l=1}^{K_i}\sum_{j=1}^{l}\mathcal{P}_{i,j}\tilde{\pi}_i[F_i(j) + a_i^j, j, l])$.

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. Therefore, we have concentrated on per class performance measures here. 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 $\lambda^* = \sum_{i=1}^{N}\lambda_i^*$.

E. 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 III-B. Specifically, we use the single class solution technique we derived in [17] 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 Appendix A for completeness.

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 III-B) and thus "difficult" to solve using a "direct" approach[4].

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 [18]. The basic approach to computing the steady state probabilities of the Markov process and the corresponding performance measures is as follows. First, we construct an upper bound model, $\mathcal{M}_i^u$, for the original Markov

[4]We could consider finite versions of the model or truncation of the infinite version [12]; 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.

process $\mathcal{M}_i$, while trying to satisfy the criteria that the new model will: (1) provide (hopefully a tight) upper bound on the desired performance measures and (2) be a "simpler"model to solve. Therefore, the upper bound model transitions that replace the original transition can be specified as follows: In Tables I and II we replace $(k-1, min(j, l-1), l-1)$ with $(k-1, 1, l-1)$, where the transition rate is $j\mu_i$. Next, we partition the state space of the original Markov process $\mathcal{M}_i$[5] 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 [3].) 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 Appendix for a detailed derivation of the solution of $\mathcal{M}_i$.)

IV. Convergence of the Iterative Method

As described in Section III, we break up the original model $\mathcal{M}$ into N single class Markovian sub-models, i.e., $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_N$. We use a set of blocking probabilities to describe the interaction among these N Markovian sub-models. In particular, the blocking probabilities $\boldsymbol{P}_i$ for class i are functions of all steady state probabilities except the steady state probability of class i, i.e., $\tilde{\boldsymbol{\pi}}_i$. Therefore, we could use N groups of simultaneous equations to represent those N Markovian sub-models since the blocking probabilities could be represented by the function of those steady state probabilities. Note that for each class i the number of unknown variables is $|\tilde{\boldsymbol{\pi}}_i|$ and there are $\sum_{i=0}^{N} |\tilde{\boldsymbol{\pi}}_i|$ variables for the whole system.

Let $\mathcal{A}_i$ be the transition rate matrix for the class i except all the entries in the first column are equal to 1. In addition, b_i is a row vector, in which all the elements are 0 but the first element is 1, and its size is the same as $|\tilde{\boldsymbol{\pi}}_i|$. Thus, we can use the following non-homogeneous system of (non-linear) equations to represent the class i Markovian sub-model $\mathcal{M}_i$:

$$\tilde{\boldsymbol{\pi}}_i \mathcal{A}_i = b_i.$$

To use the iterative method to solve N groups of simultaneous equations, we split the matrix $\mathcal{A}_i$ into two sub-matrices $\mathcal{A}_{i,n}$ and $\mathcal{A}_{i,b}$, where $\mathcal{A}_{i,n}$ is the initial nonsingular, non-blocking matrix (with all blocking probabilities are equal to 0) and $\mathcal{A}_{i,b} = \mathcal{A}_i - \mathcal{A}_{i,n}$. Given a splitting with nonsingular $\mathcal{A}_{i,n}$, we have

$$\begin{aligned} \tilde{\boldsymbol{\pi}}_i(\mathcal{A}_{i,n} + \mathcal{A}_{i,b}) &= b_i, \\ \tilde{\boldsymbol{\pi}}_i &= b_i \mathcal{A}_{i,n}^{-1} - \tilde{\boldsymbol{\pi}}_i \mathcal{A}_{i,b} \mathcal{A}_{i,n}^{-1} \end{aligned} \tag{18}$$

[5]For simplicity, we use $\mathcal{M}_i$ instead of $\mathcal{M}_i^u$ in the rest of this paper.

which leads to our iterative procedure.

$$\tilde{\boldsymbol{\pi}}_i^k = b_i \mathcal{A}_{i,n}^{-1} - \tilde{\boldsymbol{\pi}}_i^k \mathcal{A}_{i,b}^k \mathcal{A}_{i,n}^{-1}. \tag{19}$$

We note that $\mathcal{A}_{i,b}^k$ is updated based on the steady state probabilities $\tilde{\boldsymbol{\pi}}_j^{k-1}$, where $1 \leq j \leq N$ and $j \neq i$, in the previous $k-1$ round. From the above discussion, one can compute the error vector of class i, e.g., e_i^k after the kth round, we have:

$$e_i^0 = \tilde{\boldsymbol{\pi}}_i^0 - \tilde{\boldsymbol{\pi}}_i = b_i \mathcal{A}_{i,n}^{-1} - (b_i \mathcal{A}_{i,n}^{-1} - \tilde{\boldsymbol{\pi}}_i \mathcal{A}_{i,b} \mathcal{A}_{i,n}^{-1}) = -\tilde{\boldsymbol{\pi}} \mathcal{A}_{i,b} \mathcal{A}_{i,n}^{-1}, \tag{20}$$

$$e_i^1 = \tilde{\boldsymbol{\pi}}_i^1 - \tilde{\boldsymbol{\pi}}_i = \tilde{\boldsymbol{\pi}}_i^1 \mathcal{A}_{i,b}^1 \mathcal{A}_{i,n}^{-1} - \tilde{\boldsymbol{\pi}}_i \mathcal{A}_{i,b} \mathcal{A}_{i,n}^{-1}, \tag{21}$$

$$e_i^2 = \tilde{\boldsymbol{\pi}}_i^2 - \tilde{\boldsymbol{\pi}}_i = \tilde{\boldsymbol{\pi}}_i^2 \mathcal{A}_{i,b}^2 \mathcal{A}_{i,n}^{-1} - \tilde{\boldsymbol{\pi}}_i \mathcal{A}_{i,b} \mathcal{A}_{i,n}^{-1} \tag{22}$$

$$\vdots$$

$$e_i^k = \tilde{\boldsymbol{\pi}}_i^k - \tilde{\boldsymbol{\pi}}_i = \tilde{\boldsymbol{\pi}}_i^k \mathcal{A}_{i,b}^k \mathcal{A}_{i,n}^{-1} - \tilde{\boldsymbol{\pi}}_i \mathcal{A}_{i,b} \mathcal{A}_{i,n}^{-1}. \tag{23}$$

If our iterative method is able to converge, the sufficient condition is that e_k becomes very close to 0 after finite k iterations. Since both $\tilde{\boldsymbol{\pi}}_i^k$ and $\tilde{\boldsymbol{\pi}}_i$ are stationary probability vectors, we can see that as $\mathcal{A}_{i,b}^k$ approaches to the actual $\mathcal{A}_{i,b}$, e_i^k will approach 0. To illustrate the above argument, we give the proof of a 2-class system and show the convergence of the proposed iterative method.

Example: Consider a 2-class, 3-server threshold-based system as follows: $K = 3$, $K_1 = K_2 = 2$, F_1, R_1, F_2, a and R_2. We assume that there exits a stationary distribution for this system.
Let $\tilde{\boldsymbol{\pi}}_{i,j}$ be the steady state probability for class i with j servers and then $\tilde{\boldsymbol{\pi}}_1 = (\tilde{\boldsymbol{\pi}}_{1,1}, \tilde{\boldsymbol{\pi}}_{1,2})$, and $\tilde{\boldsymbol{\pi}}_2 = (\tilde{\boldsymbol{\pi}}_{2,1}, \tilde{\boldsymbol{\pi}}_{2,2})$. According to our method to compute the blocking probabilities, we have $\mathcal{P}_{1,1} = \tilde{\boldsymbol{\pi}}_{2,2}$ and $\mathcal{P}_{2,1} = \tilde{\boldsymbol{\pi}}_{1,2}$. Therefore, the goal is to solve two groups of non-homogeneous of linear equations, i.e., $\tilde{\boldsymbol{\pi}}_1 \mathcal{A}_1 = b_1$ and $\tilde{\boldsymbol{\pi}}_2 \mathcal{A}_2 = b_2$, where $\mathcal{P}_{1,1} = \tilde{\boldsymbol{\pi}}_{2,2}$ and $\mathcal{P}_{2,1} = \tilde{\boldsymbol{\pi}}_{1,2}$.

- **Step 0:** we set $\mathcal{P}_{1,1}^0 = 0$ and $\mathcal{P}_{2,1}^0 = 0$.

  $\mathcal{P}_{1,1}^0 = 0 < \mathcal{P}_{1,1} \rightarrow \tilde{\boldsymbol{\pi}}_{1,2}^0 > \tilde{\boldsymbol{\pi}}_{1,2} = \mathcal{P}_{2,1}$ and $\tilde{\boldsymbol{\pi}}_{1,1}^0 < \tilde{\boldsymbol{\pi}}_{1,1}$

  $\mathcal{P}_{2,1}^0 = 0 < \mathcal{P}_{2,1} \rightarrow \tilde{\boldsymbol{\pi}}_{2,2}^0 > \tilde{\boldsymbol{\pi}}_{2,2} = \mathcal{P}_{1,1}$ and $\tilde{\boldsymbol{\pi}}_{2,1}^0 < \tilde{\boldsymbol{\pi}}_{2,1}$
- **Step 1:** we set $\mathcal{P}_{1,1}^1 = \tilde{\boldsymbol{\pi}}_{2,2}^0$ and $\mathcal{P}_{2,1}^1 = \tilde{\boldsymbol{\pi}}_{1,2}^0$.

  $\mathcal{P}_{1,1}^1 > \mathcal{P}_{1,1} \rightarrow \tilde{\boldsymbol{\pi}}_{1,2}^1 < \tilde{\boldsymbol{\pi}}_{1,2} = \mathcal{P}_{2,1}$ and $\tilde{\boldsymbol{\pi}}_{1,1}^1 > \tilde{\boldsymbol{\pi}}_{1,1}$

  $\mathcal{P}_{2,1}^1 > \mathcal{P}_{2,1} \rightarrow \tilde{\boldsymbol{\pi}}_{2,2}^1 < \tilde{\boldsymbol{\pi}}_{2,2} = \mathcal{P}_{1,1}$ and $\tilde{\boldsymbol{\pi}}_{2,1}^1 > \tilde{\boldsymbol{\pi}}_{2,1}$.
- **Step 2:** we set $\mathcal{P}_{1,1}^2 = \tilde{\boldsymbol{\pi}}_{2,2}^1$ and $\mathcal{P}_{2,1}^2 = \tilde{\boldsymbol{\pi}}_{1,2}^1$.

  $\mathcal{P}_{1,1}^2 < \mathcal{P}_{1,1} \rightarrow \tilde{\boldsymbol{\pi}}_{1,2}^2 > \tilde{\boldsymbol{\pi}}_{1,2} = \mathcal{P}_{2,1}$ and $\tilde{\boldsymbol{\pi}}_{1,1}^2 < \tilde{\boldsymbol{\pi}}_{1,1}$

  $\mathcal{P}_{2,1}^2 < \mathcal{P}_{2,1} \rightarrow \tilde{\boldsymbol{\pi}}_{2,2}^2 > \tilde{\boldsymbol{\pi}}_{2,2} = \mathcal{P}_{1,1}$ and $\tilde{\boldsymbol{\pi}}_{2,1}^2 < \tilde{\boldsymbol{\pi}}_{2,1}$.

- **Step 3:** we set $\mathcal{P}_{1,1}^3 = \tilde{\boldsymbol{\pi}}_{2,2}^2$ and $\mathcal{P}_{2,1}^3 = \tilde{\boldsymbol{\pi}}_{1,2}^2$.

  $\mathcal{P}_{1,1}^3 > \mathcal{P}_{1,1} \rightarrow \tilde{\boldsymbol{\pi}}_{1,2}^3 < \tilde{\boldsymbol{\pi}}_{1,2} = \mathcal{P}_{2,1}$ and $\tilde{\boldsymbol{\pi}}_{1,1}^3 > \tilde{\boldsymbol{\pi}}_{1,1}$

  $\mathcal{P}_{2,1}^3 > \mathcal{P}_{2,1} \rightarrow \tilde{\boldsymbol{\pi}}_{2,2}^3 < \tilde{\boldsymbol{\pi}}_{2,2} = \mathcal{P}_{1,1}$ and $\tilde{\boldsymbol{\pi}}_{2,1}^3 > \tilde{\boldsymbol{\pi}}_{2,1}$.

  $\vdots$

- **Step 2k:** we set $\mathcal{P}_{1,1}^{2k} = \tilde{\boldsymbol{\pi}}_{2,2}^{2k-1}$ and $\mathcal{P}_{2,1}^{2k} = \tilde{\boldsymbol{\pi}}_{1,2}^{2k-1}$.

  $\mathcal{P}_{1,1}^{2k} < \mathcal{P}_{1,1} \rightarrow \tilde{\boldsymbol{\pi}}_{1,2}^{2k} > \tilde{\boldsymbol{\pi}}_{1,2} = \mathcal{P}_{2,1}$ and $\tilde{\boldsymbol{\pi}}_{1,1}^{2k} < \tilde{\boldsymbol{\pi}}_{1,1}$

  $\mathcal{P}_{2,1}^{2k} < \mathcal{P}_{2,1} \rightarrow \tilde{\boldsymbol{\pi}}_{2,2}^{2k} > \tilde{\boldsymbol{\pi}}_{2,2} = \mathcal{P}_{1,1}$ and $\tilde{\boldsymbol{\pi}}_{2,1}^{2k} < \tilde{\boldsymbol{\pi}}_{2,1}$.

- **Step 2k+1:** we set $\mathcal{P}_{1,1}^{2k+1} = \tilde{\boldsymbol{\pi}}_{2,2}^{2k}$ and $\mathcal{P}_{2,1}^{2k+1} = \tilde{\boldsymbol{\pi}}_{1,2}^{2k}$.

  $\mathcal{P}_{1,1}^{2k+1} > \mathcal{P}_{1,1} \rightarrow \tilde{\boldsymbol{\pi}}_{1,2}^{2k+1} < \tilde{\boldsymbol{\pi}}_{1,2} = \mathcal{P}_{2,1}$ and $\tilde{\boldsymbol{\pi}}_{1,1}^{2k+1} > \tilde{\boldsymbol{\pi}}_{1,1}$

  $\mathcal{P}_{2,1}^{2k+1} > \mathcal{P}_{2,1} \rightarrow \tilde{\boldsymbol{\pi}}_{2,2}^{2k+1} < \tilde{\boldsymbol{\pi}}_{2,2} = \mathcal{P}_{1,1}$ and $\tilde{\boldsymbol{\pi}}_{2,1}^{2k+1} > \tilde{\boldsymbol{\pi}}_{2,1}$.

Next, we would like to prove the following inequality:

$$\mathcal{P}_{i,1}^{2k} < \mathcal{P}_{i,1}^{2k+2} < \mathcal{P}_{i,1} < \mathcal{P}_{i,1}^{2k+3} < \mathcal{P}_{i,1}^{2k+1}, \text{ where } i = 1 \text{ or } 2 \text{ and } k \geq 0. \tag{24}$$

One can use the mathematical induction to prove the above inequality.

1. When $k = 0$, it is trivial to verify the inequality is correct for class 1 or class 2.
2. Assume $k = m$, the inequality holds for class 1 and class 2.
3. As $k = m + 1$, for class 1, we have $\mathcal{P}_{2,1}^{2m-1} > \mathcal{P}_{2,1}^{2m+1} \rightarrow \mathcal{P}_{1,1}^{2m} = \tilde{\boldsymbol{\pi}}_{2,2}^{2m-1} < \tilde{\boldsymbol{\pi}}_{2,2}^{2m+1} = \mathcal{P}_{1,1}^{2m+2}$; that is, we use the higher blocking probability for class 2, we get lower state probability $\tilde{\boldsymbol{\pi}}_{2,2}^{2m-1}$, which is the blocking probability of class 1 for the next round.

Similarly, we can prove another side of the inequality for class 1: $\mathcal{P}_{2,1}^{2m} < \mathcal{P}_{2,1}^{2m+2} \rightarrow \mathcal{P}_{1,1}^{2m+3} < \mathcal{P}_{1,1}^{2m+1}$.

Of course, the same approach can be used to prove the inequality for class 2.

Once we show that Equation (24) holds, it is not difficult to show that $\mathcal{A}_{i,b}^k$ approaches $\mathcal{A}_{i,b}$ as k increases, i.e., e_i^k is close to 0.

In the following, we give a proof of the convergence of the iterative method for a 2-class multi-server system.

Lemma 1: Consider a 2-class, K-server threshold-based system as follows: K, K_1, K_2, F_1, R_1, F_2, a and R_2. Assume that there exits a stationary state distribution for the system, the proposed iterative method for solving this system will converge to the steady state distribution.

Proof: Let $\tilde{\boldsymbol{\pi}}_{i,j}$ be the steady state probability for class i with allocated j servers, we have $\tilde{\boldsymbol{\pi}}_1 = (\tilde{\boldsymbol{\pi}}_{1,1}, \tilde{\boldsymbol{\pi}}_{1,2}, \cdots, \tilde{\boldsymbol{\pi}}_{1,K_1})$, and $\tilde{\boldsymbol{\pi}}_2 = (\tilde{\boldsymbol{\pi}}_{2,1}, \tilde{\boldsymbol{\pi}}_{2,2}, \cdots, \tilde{\boldsymbol{\pi}}_{2,K_2})$. According to our method to compute the blocking probabilities, for class 1 we have $\mathcal{P}_{1,1} = \cdots = \mathcal{P}_{1,K-K_2-1} = 0$ and $\mathcal{P}_{1,x} = \frac{\tilde{\boldsymbol{\pi}}_{2,K-x}}{\sum_{m=1}^{K-x} \tilde{\boldsymbol{\pi}}_{2,m}}$, where $K - K_2 \leq x \leq K - 2$. Similarly, for class 2 we get $\mathcal{P}_{2,1} = \cdots = \mathcal{P}_{2,K-K_1-1} = 0$ and $\mathcal{P}_{2,y} = \frac{\tilde{\boldsymbol{\pi}}_{1,K-y}}{\sum_{m=1}^{K-y} \tilde{\boldsymbol{\pi}}_{1,m}}$,

where $K - K_1 \leq y \leq K - 2$. Therefore, Our goal is to solve two groups of non-homogeneous of linear equations, i.e., $\tilde{\boldsymbol{\pi}}_1 \mathcal{A}_1 = b_1$ and $\tilde{\boldsymbol{\pi}}_2 \mathcal{A}_2 = b_2$, where $\mathcal{P}_{1,x}$ and $\mathcal{P}_{2,y}$ are defined above.

One can use the mathematical induction to prove the following inequalities for both classes:

$$\mathcal{P}_{1,x}^{2k} < \mathcal{P}_{1,x}^{2k+2} < \mathcal{P}_{1,x} < \mathcal{P}_{1,x}^{2k+3} < \mathcal{P}_{1,x}^{2k+1}, \ \ where \ \ K - K_2 \leq x \leq K - 2 \ \ and \ \ k \geq 0, \tag{25}$$

$$\mathcal{P}_{2,y}^{2k} < \mathcal{P}_{2,y}^{2k+2} < \mathcal{P}_{2,y} < \mathcal{P}_{2,y}^{2k+3} < \mathcal{P}_{2,y}^{2k+1}, \ \ where \ \ K - K_1 \leq y \leq K - 2 \ \ and \ \ k \geq 0. \tag{26}$$

When the inequalities of Eq.(25)-(26) hold, it is straightforward to show that $\mathcal{A}_{i,b}^k$ will approach to $\mathcal{A}_{i,b}$ as k increases, i.e., e_i^k is close to 0. ∎

For the general N-class K-server system, we note that the following inequality still holds for any blocking probability $\mathcal{P}_{i,j}$, i.e., $\mathcal{P}_{i,j}^{2k} < \mathcal{P}_{i,j}^{2k+2} < \mathcal{P}_{i,j} < \mathcal{P}_{i,j}^{2k+3} < \mathcal{P}_{i,j}^{2k+1}$, where $k \geq 0$. In other words, the above equation still holds under the operation of the convolution operator $\otimes$ and the function g defined in Section III. Due to the lack of space, we do not give the detailed steps of the proof here. We note that the *convergence rate* of our iterative method is still an open question, which needs further study and is beyond the scope of this work.

V. Numerical Examples and Validation of Approximation

In this section, we present numerical results to illustrate (a) the accuracy of our iterative methodology as compared with simulation, (b) the effect of server activation rate on performance measures, (c) the benefits of resource sharing among heterogeneous workload classes, and (d) the effects of threshold values on performance measures. Since the accuracy of our iterative methods is done through comparing with simulations, all simulation results are given with at least $95\% \pm 5\%$ confidence. In all experiments presented here, our iterative approach uses $\epsilon = 0.0000001$ (refer to Section III for details). Moreover, the computation time of our iterative methods is more than two orders of magnitude faster than that of simulations. This empirical evidence indicates our iterative method is fast and fairly accurate.

Experiment A: Accuracy of the iterative method.

To compare the accuracy of our iterative method in computing performance measures of the Markovian system $\mathcal{M}$, we also simulate $\mathcal{M}$, i.e., for the purpose of validating the proposed solution technique. We use mean response time for class i, $1 \leq i \leq N$, as our main performance metric of interest. Parameter settings for all test cases in Experiment A are listed in Table III.

For each test case under Experiment A, we assume all classes have the same service rate $\mu = 1$. Server activation rate is the same for all classes, and we vary this activation rate. In particular, we define $\mathcal{R} = \beta/\mu$ (where we drop the class notation for simplicity of exposition). Figure 3 illustrates the mean response time under test case 1 for $\mathcal{R} = 1$ and 10.0. It is easy to observe that the difference between the two results is small (e.g., in the case of Figure 3, the largest difference is $\approx 7\%$). Since such

Case 1	K = 10, K_1=K_2=K_3=4, $\alpha_1 : \alpha_2 : \alpha_3$= 1 : 3 : 6,
(for M^n)	F_1 = [4,8,12],R_1= [2,6,10],F_2 = [8,12,16],R_2=[5,9,13] F_3 = [6,10,14], R_3 = [3,7,11],
Exp. A	μ_1=μ_2=μ_3=1.0,
Case 2	K = 12, K_1=K_2=K_3=3, K_4=5; $\alpha_1 : \alpha_2 : \alpha_3 : \alpha_4$= 1 : 1 : 1 : 2,
(for M^n)	F_1 = [6,10],R_1= [4,7],F_2 = [4,8],R_2= [2,4], F_3 = [8,12],R_3= [6,9],F_4 = [5,9,13,17] R_4 = [3,6,9,12]
Exp. A	μ_1=μ_2=μ_3=μ_4=1.0,
Case 3	K = 12, K_1=K_2=K_3=3, K_4=5; $\alpha_1 : \alpha_2 : \alpha_3 : \alpha_4$= 1 : 2 : 3 : 4,
(for M^n)	F_1 = [6,10],R_1= [4,7],F_2 = [4,8],R_2= [2,4], F_3 = [8,12],R_3= [6,9],F_4 = [5,9,13,17] R_4 = [3,6,9,12]
Exp. A	μ_1=μ_2=μ_3=μ_4=1.0,
Case 4	K = 12, K_1=K_2=K_3=3, K_4=5; $\alpha_1 : \alpha_2 : \alpha_3 : \alpha_4$= 1 : 1 : 1 : 2,
(for M^a)	F_1 = [6,10],R_1= [4,7],F_2 = [4,8],R_2= [2,4], F_3 = [8,12],R_3= [6,9],F_4 = [5,9,13,17] R_4 = [3,6,9,12]
Exp. A	a_1 = [3,3], a_2= [3,3], a_3 = [3,3], a_4 = [3,3,3,3] μ_1=μ_2=μ_3=μ_4=1.0,
Case 5	K = 12, K_1=K_2=K_3=3, K_4=5; $\alpha_1 : \alpha_2 : \alpha_3 : \alpha_4$= 1 : 2 : 3 : 4,
(for M^a)	F_1 = [6,10],R_1= [4,7],F_2 = [4,8],R_2= [2,4], F_3 = [8,12],R_3= [6,9],F_4 = [5,9,13,17] R_4 = [3,6,9,12]
Exp. A & C	a_1 = [3,3], a_2= [3,3], a_3 = [3,3], a_4 = [3,3,3,3] μ_1=μ_2=μ_3=μ_4=1.0,
Case 6	K = 8, K_1=K_2=3, K_3=4, $\alpha_1 : \alpha_2 : \alpha_3$= 1 : 1 : 1,
(for M^a)	F_1 = [6,10],R_1= [4,7],F_2 = [6,10],R_2= [4,7], F_3 = [6,10,14], R_3 = [4,7,10],
Exp. B	a_1 = [3,3], a_2= [3,3], a_3 = [3,3,3], μ_1=μ_2=μ_3=1.0,
Case 7	K = 4, K_1=K_2=3, $\alpha_1 : \alpha_2$ = 2 : 3; a_1 = [3,3], a_2= [3,3]; μ_1=μ_2=1.0; $\mathcal{R} = 1$;
(for M^a)	**Configuration A:** F_1 =[4,8],R_1=[2,6],F_2 = [10,20],R_2=[5,10];
Exp. D	**Configuration B:** F_1 =[4,8],R_1=[2,6],F_2 = [20,40],R_2=[15,35];
	Configuration C: F_1 =[4,8],R_1=[2,6],F_2 = [40,80],R_2=[30,70];
Case 8	K = 4, K_1=K_2=3, $\alpha_1 : \alpha_2$ = 2 : 3; a_1 = [3,3], a_2= [3,3]; μ_1=μ_2=1.0; $\mathcal{R} = 1$;
(for M^a)	**Configuration A:** F_1 =[4,8],R_1=[2,6],F_2 = [6,9],R_2=[4,7];
Exp. D	**Configuration B:** F_1 =[20,40],R_1=[15,35],F_2 = [30,45],R_2=[25,40];
	Configuration C: F_1 =[40,80],R_1=[30,70],F_2 = [60,90],R_2=[50,80];
Case 9	K = 8, K_1=K_2=3, K_3=4, $\alpha_1 : \alpha_2 : \alpha_3$= 2 : 3 : 4,
(for M^a)	F_1 = [6,10],R_1= [4,7],F_2 = [6,10],R_2= [4,7], F_3 = [6,10,14], R_3 = [4,7,10],
Exp. E	a_1 = [3,3], a_2= [3,3], a_3 = [3,3,3], μ_1=μ_2=μ_3=1.0,

TABLE III

PARAMETER SETTINGS FOR EXPERIMENTS

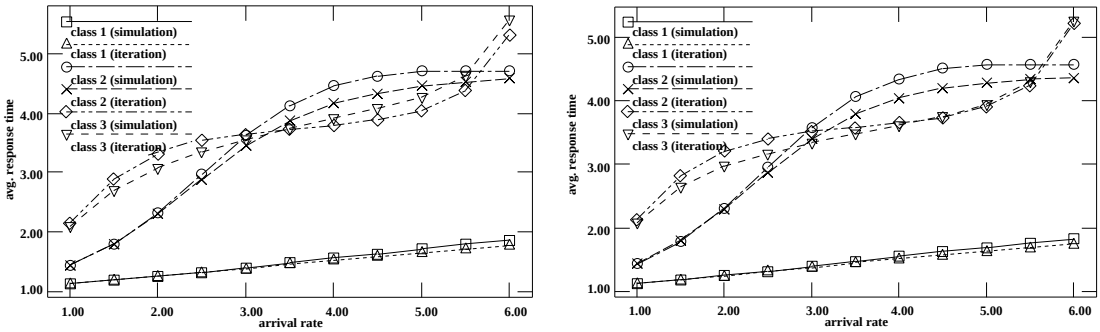
small differences are difficult to visualize using graphs, we present the remainder of the accuracy related experiments using tables.

Tables IV to VII to 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:

$$\%\text{E} = \frac{|\text{simulation result} - \text{iterative result}|}{\text{simulation result}} \text{ x } 100\% \tag{27}$$

As can be observed, in these experiments we track the performance metrics closely for all classes and the maximum error is around 10%.

As is probably expected, in our experiments, the higher error cases corresponded to fairly high con-



(a) Server activation ratio $\mathcal{R} = 1.0$ (b) Server activation ratio $\mathcal{R} = 10.0$

Fig. 3. Exp. A — Accuracy Testing (Case 1): Average response time of three classes under different server activation ratios, $\mathcal{R}$.

λ	$E[T_1]$ (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)
1.000	1.249403209	1.240894331	1.249966817	1.596548157	0.188442	0.102801	0.091916	1.082684
2.00	1.632332041	1.530185205	1.659140114	2.346862510	0.589992	1.655815	0.628414	6.931096
3.000	2.137886898	1.800536338	2.327797341	2.877125939	2.607489	4.531356	1.872697	8.398753
4.000	2.655828539	2.024466859	3.203617359	3.258738327	5.961660	7.101779	4.766918	4.467737
5.000	3.086978739	2.210009611	4.010064726	3.549132931	7.808542	8.701947	6.984125	0.872970
6.000	3.414312029	2.371654188	4.556764880	3.748743137	7.871845	8.730359	6.793586	5.205084
7.000	3.666371001	2.520343773	4.875718116	3.873368054	6.090529	8.135594	5.009943	7.823094
8.000	3.876706835	2.667619678	5.069148468	3.957965747	3.398553	6.806763	2.094356	8.864857
9.000	4.076410481	2.833209795	5.216584527	4.054224035	0.768549	5.588292	0.612172	8.653662
10.000	4.294252513	3.044849701	5.367470840	4.258024598	0.163318	6.478889	0.981928	6.684466

TABLE IV

EXP. A — ACCURACY TESTING (CASE 2): AVERAGE RESPONSE TIME FOR SERVER ACTIVATION RATIO $\mathcal{R} = 1$.

tention 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.

Note that, we have performed extensive experiments and those results are similar to the ones included here. In general, the percentage error in most testing cases was within 10% but around 14% in few cases. Next, we illustrate some of the performance tradeoffs and designs that can be studied using our technique.

λ	$E[T_1]$ (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)
1.000	1.111101334	1.240894331	1.427769462	1.596548157	0.107716	0.098113	0.223379	1.196270
2.000	1.249403209	1.530185205	2.327797341	2.346862510	0.139604	1.440456	1.633202	6.963889
3.000	1.422120871	1.800536338	3.633427552	2.877125937	0.155003	4.476466	6.016418	8.367091
4.000	1.632332041	2.024466858	4.556723717	3.258737881	0.753543	6.891397	6.901301	4.475396
5.000	1.875214383	2.210009348	4.979858567	3.549114921	1.383353	8.523428	3.532388	0.853224
6.000	2.137893813	2.371635231	5.191372533	3.748505544	3.590440	8.978094	0.609548	5.259355
7.000	2.403485514	2.519945274	5.373455187	3.871855859	5.422789	7.976693	3.374543	7.634374
8.000	2.657202108	2.664366352	5.735811514	3.952018413	7.275292	6.907009	4.165632	8.751759
9.000	2.889913738	2.822985811	7.140900686	4.037086312	8.396802	5.234919	3.636787	8.237973

TABLE V

EXP. A — ACCURACY TESTING (CASE 3): AVERAGE RESPONSE TIME FOR SERVER ACTIVATION RATIO $\mathcal{R} = 1.0$.

λ	$E[T_1]$ (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)
1.000	1.249403209	1.240894331	1.249966817	1.596548157	0.188442	0.102801	0.091916	1.082684
2.000	1.632332041	1.530185205	1.659140114	2.346862510	0.589992	1.655815	0.628414	6.931096
3.000	2.137886898	1.800536338	2.327797341	2.877125937	2.607489	4.531356	1.872697	8.398753
4.000	2.655828538	2.024466858	3.203617358	3.258737880	5.961660	7.101779	4.766918	4.467751
5.000	3.086978271	2.210009329	4.010064114	3.549114866	7.807762	8.699709	6.980227	0.879640
6.000	3.414279404	2.371633439	4.556723717	3.748504405	7.872210	8.735477	6.847014	5.201449
7.000	3.665701028	2.519889412	4.874913397	3.871844972	6.090198	8.137421	4.974583	7.772737
8.000	3.871141342	2.663543765	5.062810736	3.951957924	3.526046	7.062987	2.360242	8.736353
9.000	4.053114585	2.814672095	5.191371673	4.036856602	1.660845	6.500396	0.292736	8.240439
10.000	4.235582423	2.994503482	5.306591592	4.216504326	1.340852	7.132456	0.282897	6.970984
11.000	4.459602796	3.243621554	5.457264738	4.763969135	2.877606	8.946037	1.377628	5.284929
12.000	4.815342733	3.651148297	5.735634028	7.980613244	5.087811	9.514688	4.097968	3.596882

TABLE VI

EXP. A — ACCURACY TESTING (CASE 4): AVERAGE RESPONSE TIME FOR SERVER ACTIVATION RATIO $\mathcal{R} = 1.0$.

Experiment B: Effect of Service Activation Rate on System Performance.

In this experiment, we consider how the server activation rate may affect the average response time of different classes of customers. Let us consider test case 6 in Table III. We vary the server activation ratio $\mathcal{R}$ from 0.1 to 100. In other words, server activation rate can be ten times slower than the average service rate or up to 100 times faster than the average service rate. Table VIII illustrates the average response time for all classes of customers under different traffic loadings. As we can observe, it is important to consider server activation issues in the performance analysis of dynamic resource management systems using threshold-based techniques. In particular, when the system is operating at a low server activation ratio (i.e., when $\mathcal{R}$ is small), the average response time for all classes is higher than in situation where

λ	$E[T_1]$ (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)
1.000	1.111101334	1.240894331	1.427769462	1.596548157	0.174672	0.138768	0.014092	1.301958
2.000	1.249403209	1.530185205	2.327797341	2.346862510	0.190320	1.628630	1.589170	6.970281
3.000	1.422120871	1.800536338	3.633427552	2.877125937	0.017206	4.421477	6.187979	4.456612
4.000	1.632332041	2.024466858	4.556723717	3.258737880	0.582615	7.084183	6.795926	0.910167
5.000	1.875214307	2.210009329	4.979858554	3.549114866	1.270779	8.620992	3.584426	5.289322
6.000	2.137886898	2.371633439	5.191371673	3.748504405	3.064727	8.844618	0.579377	7.780205
7.000	2.403313249	2.519889412	5.373437048	3.871844972	4.904982	8.076828	3.464238	8.703671
8.000	2.655828538	2.663543765	5.735634028	3.951957924	6.529355	6.896232	3.719736	8.307669
9.000	2.885087046	2.814672095	7.139840165	4.036856602	8.310797	5.611460	1.103102	7.699883
9.500	2.989496161	2.899102696	10.324209470	4.105645593	9.692825	5.219340	2.013128	7.569769

TABLE VII

EXP. A — ACCURACY TESTING (CASE 5): AVERAGE RESPONSE TIME FOR SERVER ACTIVATION RATIO $\mathcal{R} = 1.0$.

the system is operating at high server activation ratios. The performance difference is more prominent when we have moderate to high traffic loadings.

Experiment C: Benefits of Resource Sharing.

In this experiment, we illustrate the benefits of resource sharing. In particular, we consider the case wherein the number of common resources, K, is less than the peak resource demand $\sum_{i=1}^{N} K_i$. The reasons for using fewer resources are to reduce the overall system operational cost and to take advantage of the fact that the probability of all classes being at their high workload demands simultaneously might be quite low. Of course, one needs to quantify the above advantages, especially when hysteresis control is used in threshold-based system. We consider test case 5 in Table III. In this test case, the total peak resource demand is $\sum K_i = 14$. We consider two cases, where we set the values of K, the total amount of system resources, to be 11 and 14, respectively. Note that when $K = 14$, we will not have resource contention among classes. For this experiment, we assume a homogeneous server activation ratio $\mathcal{R} = \beta/\mu = 1.0$. Figure 4 illustrates the average response time of three classes of workload, under both cases. As we can observe, the *difference* in the average response time is *not* significant, i.e., the maximum percentage error between cases $K = 11$ and $K = 14$ is 6%. This implies that, in this experiment, one can use fewer resources, with resource saving of around 21%, while at the same time providing *comparable* performance (i.e., comparable to the case where the amount of resources is equal to the peak workload demands of all classes).

Experiment D: Effects of Threshold Settings.

As stated in Section I, we believe that our solution technique for this multi-class, multi-server threshold-based queueing system with hysteresis behavior can be used as an efficient tool for searching for good threshold values. In this experiment, we illustrate that this is an important issue by showing that different

λ	$\mathcal{R}$ activation ratio	$E[T_1]$ (iterative)	$E[T_2]$ (iterative)	$E[T_3]$ (iterative)
0.900	0.100	1.425080947	1.425080947	1.425071989
0.900	1.000	1.422120871	1.422120871	1.422119765
0.900	10.00	1.421043244	1.421043244	1.421042825
0.900	100.0	1.420915968	1.420915968	1.420915582
3.600	0.100	5.379141763	5.379141763	5.393195375
3.600	1.000	3.414267196	3.414267196	3.387999226
3.600	10.00	3.145905856	3.145905856	3.128791883
3.600	100.0	3.118374248	3.118374248	3.101954561
5.400	0.100	7.299335678	7.299335678	7.223295397
5.400	1.000	4.050733764	4.050733764	3.927137459
5.400	10.00	3.587663185	3.587663185	3.465490149
5.400	100.0	3.539729932	3.539729932	3.418567750
8.100	0.100	10.616787909	10.616787909	7.852223350
8.100	1.000	6.283156018	6.283156018	4.297965780
8.100	10.00	5.764497387	5.764497387	3.741218408
8.100	100.0	5.709881527	5.709881527	3.682093400

TABLE VIII

EXP. B — SERVICE ACTIVATION RATE EXPERIMENT (CASE 6): AVERAGE RESPONSE TIME FOR ALL CLASSES UNDER DIFFERENT SERVER ACTIVATION RATIOS AND TRAFFIC LOADINGS.

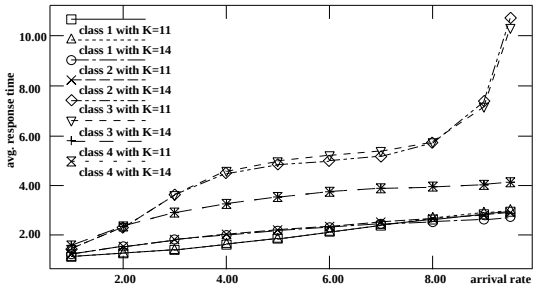
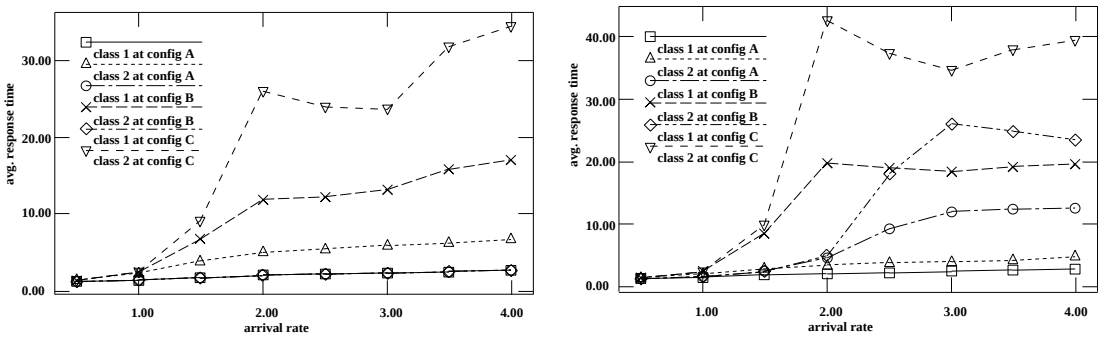


Fig. 4. Exp. C — Resource Sharing Experiment (Case 5): Average response time for four classes of workload with $K = 11$ and 14.

threshold values can result in vastly different performance measures. Since our solution can quickly generate corresponding performance results, it should be a good tool for exploring proper threshold settings. In particular, we consider a four servers systems with two classes whose parameter settings are listed in Table III. Note that, in each test case we have three configurations A, B, and C where in each configuration has different threshold settings. Figure 5 depicts the corresponding expected response time results, which illustrate that changes in threshold values result in significant changes in the expected response time of the different classes. Therefore, one can use the proposed solution technique to search for the proper threshold values such that the expected response time of the different workloads is satisfactory, within certain levels of system loadings.



(a) Case 7: mean resp. time vs. diff. $\boldsymbol{F}_i$ and $\boldsymbol{R}_i$ (b) Case 8: mean resp. time vs. diff. $\boldsymbol{F}_i$ and $\boldsymbol{R}_i$

Fig. 5. Exp. D — Threshold Setting Experiment (Case 7 & Case 8): Average response time for two workload classes under different thresholds $\boldsymbol{F}_i$ and $\boldsymbol{R}_i$.

Experiment E: Instantaneous Model vs. Non-Instantaneous Model.

The motivation for this experiment is to study the difference between the instantaneous model and the non-instantaneous model and when is important to take the server activation ratio into consideration for design an efficient multi-class multi-server system.

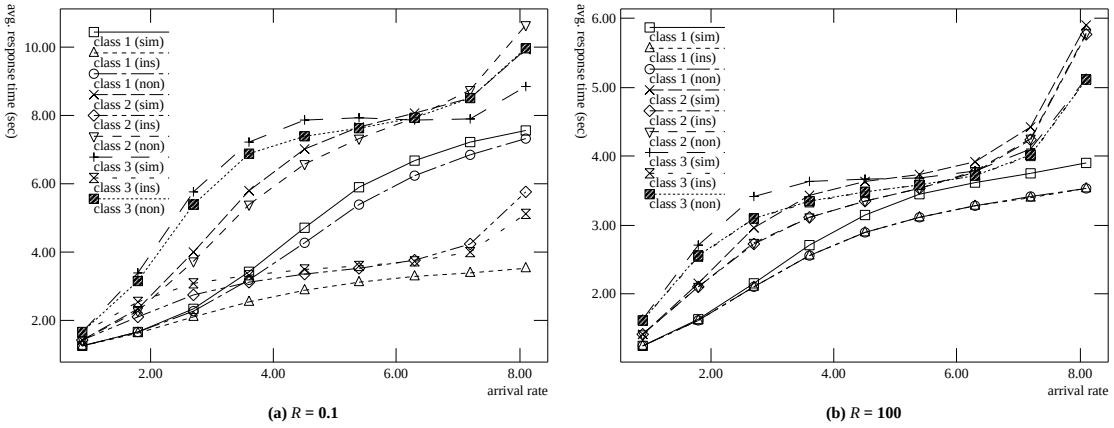


Fig. 6. Exp. E — Instantaneous Model vs. Non-Instantaneous Model (Case 9).

In Figure 6(a), the results show that as the activation ratio is smaller than the service rate, the system performance is mostly dominated by the server activation time. This is why the curve of the instantaneous model deviates from the simulation curve a lot (e.g., for class 1 the largest difference is $\approx 100\%$) while the curve of the non-instantaneous model is close to the simulation curve. That is, it is important to consider the server activation ratio in order to construct an precisely analytic model for the system. On the other hand, as illustrated in Figure 6(b), when the service rate is much faster than the server activation rate, both the results of instantaneous and non-instantaneous models are similar with that of the simulation. In other words, the instantaneous model is a good approximation one as the server activation ratio is large.

VI. CONCLUSIONS

In this paper we considered efficient and accurate computation of performance metrics for a system which shares K servers (or resources) among N heterogeneous classes of workloads, where server allocation and de-allocation is dictated by a class specific threshold-based policy with hysteresis control. An important and distinguishing characteristic of our work, as compared to previous efforts, is that we consider the modeling and analysis of a multi-class system with *non-instantaneous* server activation, which is of use in studying performance characteristics of many important applications. We presented an efficient iterative approximation technique for solving such models and illustrated through numerical results that this technique is reasonably accurate (in the presented experiments the deviation from the exact solution is within $\approx 10\%$) and fast (with more than two orders of magnitude improvement in computation time compared to simulations). Moreover, our numerical results also illustrated that (a) server activation characteristics have a significant effect on the system's performance, (b) dynamic resource sharing, through the use of threshold-based techniques, can result in significant cost savings (i.e., through sharing of a pool of resources among heterogeneous workload classes) without detrimental effects on the class' performance, and (c) proper threshold settings can have a significant effect on the class' performance characteristics. Consequently, all these results indicate that efficient solution techniques for models of such dynamic resource management systems are critical for proper design and performance studies of these systems. And, we believe that the technique presented in this paper is one such approach which will lead to better system designs.

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.S. 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.S. Lui. A fast and accurate iterative solution of a multi-class threshold-based queueing system with hysteresis. In *sigmetrics2000*, pages 196–206, CA, USA, June 2000.

[5] Leana Golubchik and John C.S. Lui. Bounding of performance measures for threshold-based queuing systems: Theory and application to dynamic resource management in video-on-demand servers. *IEEE Transactions on Computers*, 51(4):353–372, 2002.

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

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

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

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

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

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

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

[13] L. Kleinrock. *Queueing Systems, Volume I*. Wiley-Interscience, 1975.

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

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

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

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

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

[19] 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.

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

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

[22] 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.

[23] M. F. Neuts. *Matrix-geometric Solutions in Stochastic Models – an Algorithmic Approach*. John Hopkins University Press, Baltimore, MD, 1960.

[24] William J. Stewart. *Introduction to Numerical Solution of Markov Chains*. Princeton University Press, 1994.

APPENDIX A: ANALYSIS OF A SINGLE CLASS MODEL

In this appendix, we address the analysis of a single class model. Due to lack of space, we only present the derivation for model $\mathcal{M}^a$. The derivation for model $\mathcal{M}^b$ is similar to the derivation for the *last level*, K_i of model $\mathcal{M}^a$.

The goal here is to compute the steady state probabilities $\tilde{\boldsymbol{\pi}}_i[k,j,l]$ for all $(k,j,l) \in \mathcal{S}_i$, where $\mathcal{S}_i$ is the state space of the Markov process $\mathcal{M}_i$ (see Section III-B). Note that, since this computation must be performed in each iteration of the procedure given in Section III, for clarity of presentation in this appendix, we will omit from the notation indication of the iteration step number. As stated in Section III-E, the first step is to partition the state space. Specifically, given the original Markov process $\mathcal{M}_i$, let us partition the state space $\mathcal{S}_i$ into K_i disjoint sets $\mathcal{S}_i^l$, where:

$$\mathcal{S}_i^l \quad = \quad \{(k,j,l) \mid (k,j,l) \in \mathcal{S}_i \text{ and } j \leq l\} \text{for } l = 1, 2, \ldots, K_i.$$

We can view partition $\mathcal{S}_i^l$ as representing all states corresponding to exactly l target allocated servers. Let us consider the analysis of $\mathcal{S}_i^l$ where $2 \leq l \leq K_i - 1$. We first define another Markov process $\mathcal{M}_i^l$

with a corresponding steady state probability vector $\boldsymbol{\pi}_i^l$, for $l \in \{2, \ldots, K_i - 1\}$, such that the state space of $\mathcal{M}_i^l$ corresponds to the states in $\mathcal{S}_i^l$. For ease of notation, we will use $\pi_i^l(k, j, l)$ to refer to the steady state probability of state $(k, j, l) \in \mathcal{S}_i^l$ (since all states in $\mathcal{S}_i^l$ have the same l, the target allocated servers to class i). The transition structure of $\mathcal{M}_i^l$ is similar to the transition structure of $\mathcal{M}_i$ for the states in $\mathcal{S}_i^l$, except for the following modifications:

step 1: a transition from $(R_i(l-1)+1, j, l)$ to $(R_i(l-1), j, l-1)$ or $(R_i(l-1), j-1, l-1)$ in the original process $\mathcal{M}_i$ is replaced by transitions from $(R_i(l-1)+1, j, l)$ to the states $(F_i(l-1)+1+m, j, l)$ in $\mathcal{M}_i^l$, where $0 \leq m \leq a_i^{l-1}$, each at the rate of $r_i^{l-1}(m) l \mu_i$ where

$$r_i^{l-1}(m) = \frac{\pi_i^{l-1}(F_i(l-1)+m, j, l-1)}{\sum_{z=0}^{a_i^{l-1}} \pi_i^{l-1}(F_i(l-1)+z, j, l-1)}$$

and $\pi_i^{l-1}(F_i(l-1)+m, j, l-1)$ is the conditional steady state probability corresponding to the state with $F_i(l-1)+m$ customers, j busy servers and $l-1$ target allocated servers, conditioned on being in $\mathcal{S}_i^{l-1}$. Note that this conditional steady state probability is obtained by solving the Markovian model $\mathcal{M}_i^{l-1}$ (see Theorem 1 below and Section A for details).

step 2: a transition from $(F_i(l)+m, j, l)$ to $(F_i(l)+m+1, j, l+1)$ in the original process $\mathcal{M}_i$, is replaced by a transition from $(F_i(l)+m, j, l)$ to $(R_i(l), 1, l)$ in $\mathcal{M}_i^l$, $0 \leq m \leq a_i^l$, each at the rate of $\lambda_i(1 - \mathcal{P}_{i,l})$.

Similarly, for the first level $l = 1$, we can order the states in $\mathcal{S}_i^1$ as follows:

$$\{(0,1,1), \ldots, (R_i(1),1,1), \ldots, (F_i(1),1,1), \ldots (F_i(1)+a_i^1, 1, 1)\}$$

and then define the Markov process $\mathcal{M}_i^1$ such that the state space of $\mathcal{M}_i^1$ corresponds to the states in $\mathcal{S}_i^1$. The transition structure of $\mathcal{M}_i^1$ is similar to that of $\mathcal{M}_i$ for the states in $\mathcal{S}_i^1$, except that a transition from $(F_i(1)+m, 1, 1)$ to $(F_i(1)+1+m, 1, 2)$ in $\mathcal{M}_i$ is replaced by a transition from $(F_i(1)+m, 1, 1)$ to $(R_i(1), 1, 1)$ in $\mathcal{M}_i^1$, where $0 \leq m \leq a_i^1$, each at the rate of $\lambda_i(1 - \mathcal{P}_{i,1})$.

Finally, for the last level $l = K_i$, define the Markov process $\mathcal{M}_i^{K_i}$ such that the state space of $\mathcal{M}_i^{K_i}$ corresponds to the states in $\mathcal{S}_i^{K_i}$. The transition structure of $\mathcal{M}_i^{K_i}$ is similar to that of $\mathcal{M}_i$ for the states in $\mathcal{S}_i^{K_i}$, except that a transition from $(R_i(K_i-1)+1, j, K_i)$ to $(R_i(K_i-1), j, K_i-1)$ or $(R_i(K_i-1), j-1, K_i-1)$ in $\mathcal{M}_i$ is replaced by transitions from $(R_i(K_i-1)+1, j, K_i)$ to the states $(F_i(K_i-1)+1+m, j, K_i)$ in $\mathcal{M}_i^{K_i}$, where $0 \leq m \leq a_i^{K_i-1}$, each at the rate of $r_i^{K_i-1}(m) l \mu_i$, where

$$r_i^{K_i-1}(m) = \frac{\pi_i^{K_i-1}(F_i(K_i-1)+m, j, K_i-1)}{\sum_{z=0}^{a_i^{K_i-1}} \pi_i^{K_i-1}(F_i(K_i-1)+z, j, K_i-1)}$$

and $\pi_i^{K_i-1}(F_i(K_i-1)+m, j, K_i-1)$ is the conditional steady state probability corresponding to the state with $F_i(K_i-1)+m$ customers, j busy servers and K_i-1 target allocated servers, conditioned on

being in $\mathcal{S}_i^{K_i-1}$. Note that this conditional steady state probability is obtained by solving the Markovian model $\mathcal{M}_i^{K_i-1}$ (see Theorem 1 below and Section A for details).

Lastly, we state the following theorem which is analogous to the one given in [17] for a single class model. It is needed to show the relationship between the solution of $\mathcal{M}_i^l$ and $\mathcal{M}_i$. (Note that in this appendix we use the same notation, namely $\boldsymbol{\pi}_i^l$, when referring to the steady state probability vector of $\mathcal{M}_i^l$ and when referring to the conditional steady state probability vector of $\mathcal{M}_i$, conditioned on being in the set $\mathcal{S}_i^l$.) The proof of this theorem is also analogous to the one given in [17] and so we do not repeat the details here.

Theorem 1: The steady state probabilities solution of the Markov process $\mathcal{M}_i^l$ is the conditional steady state probabilities solution for the states in $\mathcal{S}_i^l$ of the original Markov process $\mathcal{M}_i$, given that the system is in partition $\mathcal{S}_i^l$.

A. *Analysis of* $\mathcal{M}_i^l$

Let us now describe the computational procedure for obtaining the steady state probability vector for each Markov process $\mathcal{M}_i^l$, $l = 1, 2, \ldots, K_i$. Let $\boldsymbol{Q}_i^l$ and $\boldsymbol{\pi}_i^l$ be the transition rate matrix and the steady state probability vector of $\mathcal{M}_i^l$, respectively. Since the state space of $\mathcal{M}_i^l$, $l = 1, 2, \ldots, K_i - 1$, is finite and (given a well designed system) small, we can obtain the steady state probability vector by solving the following system of linear equations [24]

$$\boldsymbol{\pi}_i^l \boldsymbol{Q}_i^l = \mathbf{0} \quad ; \quad \boldsymbol{\pi}_i^l \boldsymbol{e} = 1$$

where $\boldsymbol{e}$ is a column vector of 1's. To improve the complexity of computing $\boldsymbol{\pi}_i^l$, $l = 1, 2, ..., K_i - 1$, we can use well-known numerical methods [24], which for instance, can take advantage of the (potential) sparsity of $\boldsymbol{Q}_i^l$.

What remains is the computation of $\boldsymbol{\pi}_i^{K_i}$. Note that the state space of $\mathcal{M}_i^{K_i}$ is infinite; therefore, we cannot use, for instance, a direct method [24] for solving this system of linear equations. Instead, we can compute $\boldsymbol{\pi}_i^{K_i}$ by using Matrix-analytic methods[23]. Let us define a boundary set $\mathcal{S}_B$ (which is the boundary portion under the matrix-analytic notation) with state space

$$\{(R_i(K_i-1)+1, j, K_i), \ldots, (F_i(K_i-1), j, K_i), \ldots, (F_i(K_i-1)+a_i^{K_i-1}+1, j, K_i)\}$$

such that transitions between these states are identical to those between the corresponding states in $\mathcal{M}_i^{K_i}$. Similarly, we define another set $\mathcal{S}_I$ (which is the repetitive portion under the matrix-analytic notation) with an infinite state space

$$\{(F_i(K_i-1)+a_i^{K_i-1}+2, j, K_i), (F_i(K_i-1)+a_i^{K_i-1}+3, j, K_i), \ldots\}$$

and transitions between states identical to those between the corresponding states in $\mathcal{M}_i^{K_i}$. It is not difficult to observe that the transition structure in $\mathcal{S}_I$ is *repetitive*. Therefore, given these two sets of states, $\mathcal{S}_B$ and $\mathcal{S}_I$, one has an efficient algorithm [23] for computing the steady state probability vector $\boldsymbol{\pi}_i^{K_i}$.

B. Analysis of the Aggregated Process for Class i

Once we have obtained an expression for the steady state probability vector of each $\mathcal{M}_i^l$, which is also the conditional steady state probability vector of $\mathcal{M}_i$, given that the system is in $\mathcal{S}_i^l$, the only remaining step (as outlined in Section III-E) is to find the aggregate state probability of the system being in $\mathcal{S}_i^l$. Therefore, for each l, $1 \leq l \leq K_i$ let us aggregate all the states in $\mathcal{S}_i^l$ into a single state. The transition state diagram of the resulting aggregated process is illustrated in Figure 7. The transition rates of the

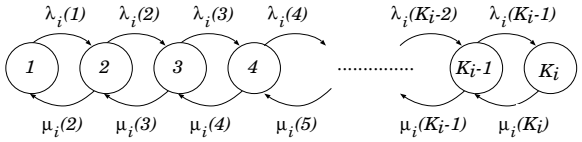


Fig. 7. State transition diagram for aggregated process for class i.

aggregated process can be computed as follows:

$$\lambda_i(l) = \lambda_i(1-\mathcal{P}_{i,l}) \sum_{j=1}^{l} \sum_{k=F_i(l)}^{F_i(l)+a_i^l} \pi_i^l(k,j,l) \quad l = 1,2,\ldots,K_i-1 \tag{28}$$

$$\mu_i(l) = j\mu_i \sum_{j=1} l\pi_i^l(R_i(l-1)+1,j,l) \quad l = 2,3,\ldots,K_i \tag{29}$$

where $\pi_i^l(k,j,l), k = F_i(l),\ldots,F_i(l)+a_i^l$, and $\pi_l^i(R_i(l-1)+1,j,l)$ are the conditional steady state probabilities obtained in Section A.1. The steady state probability vector, $\boldsymbol{\pi}_i^*$, of this aggregated process is computed as follows [13]:

$$\pi_i^*(1) = \left[1+\sum_{l=2}^{K_i}\prod_{j=1}^{l-1}\left(\frac{\lambda_i(j)}{\mu_i(j+1)}\right)\right]^{-1} \tag{30}$$

$$\pi_i^*(m) = \left[1+\sum_{l=2}^{K_i}\prod_{j=1}^{l-1}\left(\frac{\lambda_i(j)}{\mu_i(j+1)}\right)\right]^{-1} \prod_{j=1}^{m-1}\left[\frac{\lambda_i(j)}{\mu_i(j+1)}\right] \quad m = 2,3,\ldots,K_i \tag{31}$$

C. Performance Measures for Class i

At this point we have all the necessary information to compute the steady probabilities for $\mathcal{M}_i$. That is, once we determine, for each l: (1) the conditional state probabilities of all states in $\mathcal{S}_i^l$, given that the

system is in $\mathcal{S}_i^l$ and (2) the steady state probability of being in state l of the aggregated process, then the steady state probability of each individual state (k, j, l) in $\mathcal{M}_i$ can be expressed as:

$$\tilde{\pi}_i[k, j, l] \quad = \quad \pi_i^j(k, j, l)\pi_i^*(l) \quad \text{where } (k, j, l) \in \mathcal{S}_i^l. \tag{32}$$

Then (as outlined in Section III-E) we can compute various performance measures; more specifically, we can compute many performance measures which can be expressed in the form of a Markov reward function, $\mathcal{R}_i$, where $\mathcal{R}_i = \sum_{k,j,l} \tilde{\pi}_i[k, j, l]R_i(k, j, l)$ and $R_i(k, j, l)$ is the reward for state (k, j, l) of class i. Two useful performance measures for our system are the expected number of customers and the expected response time for class i, $i = 1, 2, \ldots, N$. Below, we illustrate how easy it is to obtain such performance measures, once we have the steady state probabilities; for instance, the expected number of customers can be expressed as a Markov reward functions, where $R_i(k, j, l) = k$.

Let $\bar{N}_i$ and $\bar{T}_i$ denote the expected number of customers and the expected response time, respectively, of class i model, corresponding to the Markov process $\mathcal{M}_i$. Then $\bar{N}_i$ can be expressed as:

$$\bar{N}_i \quad = \quad \sum_{k=1}^{F_i(1)+a_i^1} k\tilde{\pi}_i[k, 1, 1] + \sum_{l=2}^{K_i-1} \sum_{j=1}^{l} \sum_{k=R_i(j-1)+1}^{F_i(j)+a_i^j} k\tilde{\pi}_i[k, j, l] + \sum_{j=1}^{K_i} \sum_{k=R_i(K_i-1)+1}^{\infty} k\tilde{\pi}_i[k, j, K_i]. \tag{33}$$

Using Little's result [16], we have:

$$\bar{T}_i \quad = \quad \left[\lambda_i \left(1 - \sum_{l=1}^{K_i} \sum_{j=1}^{l} \mathcal{P}_{i,j}\tilde{\pi}_i[F_i(j) + a_i^j, j, l]\right)\right]^{-1} * \bar{N}_i. \tag{34}$$