Optimal and Structured Call Admission Control Policies for Resource-Sharing Systems

Jian Ni, *Student Member, IEEE*, Danny H. K. Tsang, *Senior Member, IEEE*, Sekhar Tatikonda, *Member, IEEE*, and Brahim Bensaou, *Member, IEEE*

***Abstract*—Many communication and networking systems can be modeled as resource-sharing systems with multiple classes of calls. Call admission control (CAC) is an essential component of such systems. Markov decision process (MDP) tools can be applied to analyze and compute the optimal CAC policy that optimizes certain performance metrics of the system. But for most practical systems, it is prohibitively difficult to compute the optimal CAC policy using any MDP algorithm because of the "curse of dimensionality." We are, therefore, motivated to consider two families of structured CAC policies: reservation and threshold policies. These policies are easy to implement and have good performance in practice. However, since the number of structured policies grows exponentially with the number of call classes and the capacity of the system, finding the optimal structured policy is a complex unsolved problem. In this paper, we develop fast and efficient search algorithms to determine the parameters of the structured policies. We prove the convergence of the algorithms. Through extensive numerical experiments, we show that the search algorithms converge quickly and work for systems with large capacity and many call classes. In addition, the returned structured policies have optimal or near-optimal performance, and outperform those structured policies with parameters chosen based on simple heuristics.**

***Index Terms*—Call admission control (CAC), combinatorial optimization, Markov decision process (MDP), reservation policy, resource sharing, threshold policy.**

I. Introduction

We consider a general model of resource-sharing systems with multiple classes of calls (in this paper the term *call* is used for all types of service). The system has C units of resources, and is shared by K different classes of calls. Class-k calls arrive according to a Poisson process with rate λ_k, independent of the call arrival processes of other classes. When a call arrives, the system either accepts or blocks the call based on a certain *call admission control* (CAC) policy. A class-k call, once accepted, occupies b_k units of resources, and the b_k units of resources will be released simultaneously when the call finishes after an exponentially distributed service time with mean $1/\mu_k$ (extensions to general service time distributions will also be discussed). The system is a loss system with no queueing. This model is also referred to as a *stochastic knapsack* [8], [20], [22].

Paper approved by T. T. Lee, the Editor for Wireless Communication Theory of the IEEE Communications Society. Manuscript received November 13, 2004; revised March 30, 2006. This paper was presented in part at the IEEE Infocom, Miami, FL, March 2005.

J. Ni and S. Tatikonda are with the Department of Electrical Engineering, Yale University, New Haven, CT 06520 USA (e-mail: jian.ni@yale.edu; sekhar.tatikonda@yale.edu).

D. H. K. Tsang is with the Department of Electrical and Electronic Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong (e-mail: eetsang@ee.ust.hk).

B. Bensaou is with the Department of Computer Science, the Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong (e-mail: brahim@cs.ust.hk).

Digital Object Identifier 10.1109/TCOMM.2006.887498

Many communication and networking systems can be modeled as resource-sharing systems. CAC is an essential component of these systems. It determines how the system resources should be shared among the different call classes in order to achieve good performance.

Example 1: In circuit-switched networks (or virtual circuit-switched networks), an access link with certain amount of bandwidth needs to accommodate different types of service including data, audio, video, and other emerging multimedia applications, each with heterogenous bandwidth requirement and traffic statistics. The goal of CAC here is, for example, to maximize the throughput of the link.

Example 2: In mobile cellular systems, at each cell, a base station with a limited number of channels (frequency bands, time slots, or codes) needs to handle both new calls originating at this cell and handoff calls from neighboring cells. Normally, handoff calls have higher priority over new calls because one does not want to terminate an ongoing call. The goal of CAC here is, for example, to minimize the new/handoff call blocking probabilities while giving priority to handoff calls.

Example 3: In multimedia content delivery networks (CDNs), a CDN server with certain amount of streaming bandwidth needs to serve clients who are requesting for different types of multimedia objects that have different session times and streaming bandwidth requirements. The CDN server collects a revenue (the amount is determined by the type of the multimedia object) from each accepted client. The goal of CAC here is, for example, to maximize the revenue rate of the CDN server.

This resource-sharing model also has numerous instances in many computer and social systems [7], [15]. The simplest CAC policy, known as *complete sharing* (CS), is to accept a call whenever the system has sufficient resources, and to block the call otherwise. The CS policy is easy to implement, and performs well if the traffic demands are light or there is only one call class. However, in today's communication and networking systems, traffic demands are unpredictable, and most often, the system operates close to its full-capacity regime, thus, the CS policy may lead to poor resource use. In addition, *service differentiation* is now becoming a more and more desirable feature for many systems. For example, some users may want to pay more to get better quality of service. If each accepted call contributes a class-based revenue, then the CS policy may lead to a poor revenue rate for the system. Therefore, it is desirable to implement a CAC policy that optimizes certain performance metrics, such as the call-blocking probabilities, throughput, or revenue rate of the system (these performance metrics are defined in the long term or at the steady state).

A resource-sharing system is said to be *single service* if calls of all classes have the same resource requirement and mean service time. These include traditional telephone networks (in which each established call occupies one circuit) and mobile cellular systems (in which each accepted call uses one channel). Due to service differentiation, the system may charge differently among the classes so that each accepted call contributes a class-based revenue. It is well known that the optimal CAC policy which maximizes the system revenue rate has a simple structure [3], [14], [15], which we call a *reservation* policy. The reservation policies are also known as *trunk reservation* in the literature of telephone/circuit-switched networks [2], [11], [17], and *cutoff priority scheme/guard channel* in the literature of mobile cellular systems [9], [13], [19].

For single-service systems, the system state under a reservation policy can be modeled using a one-dimensional (1-D) birth–death process. Direct calculation of its steady-state probabilities may face numerical overflow/underflow problems. In this paper, we develop a set of fast recursive formulas that can evaluate the performance of any reservation policy for single-service systems with very large capacity and an arbitrary number of call classes. However, since the number of reservation policies grows exponentially with the number of call classes and the system capacity, finding the optimal reservation policy (for single-service systems, this is also the optimal CAC policy) is a complex combinatorial optimization problem. We propose an *iterative coordinate search algorithm* to find an *ordered coordinate optimal reservation policy* among all reservation policies. We prove the convergence of the algorithm. Through extensive numerical experiments, we show that the algorithm converges quickly and the returned reservation policy has optimal or near-optimal performance.

Most communication and networking systems have evolved into, or are expected to evolve into, *multiservice* systems which can support both narrowband and wideband multimedia services. In multiservice resource-sharing systems, calls of different classes may have heterogenous resource requirements and mean service times. For a multiservice system, the optimal CAC policy does not have a simple structure, and for most systems of practical interest, it is prohibitively difficult to compute the optimal CAC policy because of the "curse of dimensionality" [3], [21]. In addition, a multidimensional birth–death process is required to model the system state under a reservation policy, which requires excessive computational effort to evaluate the system performance. This makes the task of finding the optimal reservation policy extremely difficult.

Another family of structured CAC policies, the *threshold* policies [7], [8], [20], [22], have been proposed and extensively studied for resource-sharing systems/stochastic knapsacks. Under a threshold policy, the stationary distribution of the system state has a product form [20], and the system performance can be evaluated through efficient convolution algorithms [24]. However, since the number of threshold policies grows exponentially with the number of call classes and the system capacity, finding the optimal threshold policy is a complex combinatorial optimization problem. Again, our proposed iterative coordinate search algorithm can be applied to find a *coordinate optimal threshold policy* among all threshold policies. Through extensive numerical experiments, we show that the algorithm converges quickly and the returned threshold policy has optimal or near-optimal performance.

We organize the paper as follows. In Section II, we introduce the system model and the optimization problem. In Sections III and IV, we consider CAC for single-service and multiservice resource-sharing systems, respectively. Fast and efficient search algorithms are developed to determine the parameters of the reservation policy (for single-service systems) and the threshold policy, with the objective of maximizing the system revenue rate. In both sections, we evaluate the performance of the search algorithms in terms of their convergence rate and the quality of the returned structured policies through extensive numerical experiments. In Section V, we compare reservation and threshold policies by their generated revenue rate, fairness, and robustness. We conclude the paper in Section VI.

II. The Model and the Optimization Problem

Let $\mathbf{n} = (n_1, n_2, \ldots, n_K)$ denote the state of the system, where $n_k \in \mathbb{Z}^+$ (the set of nonnegative integers) is the number of class-k calls that are currently in the system. Let $\Omega_{\text{cs}} = \{\mathbf{n} : \mathbf{n} \cdot \mathbf{b}^T \triangleq \sum_{k=1}^{K} n_k b_k \leq C\}$ be the set of all feasible system states, where $\mathbf{b} = (b_1, b_2, \ldots, b_K)$. The *admissible state set* of a CAC policy, which is a subset of Ω_{cs}, includes all states the system may enter under that policy.

Let B_k be the *blocking probability* of class-k calls. The *system revenue rate* g is defined as the expected revenue collected by all ongoing calls in the system per unit time. There are two revenue-collecting models. In the first model, when a class-k call is accepted, a lump-sum revenue R_k is collected. In the second model, a class-k call generates revenue at rate R'_k per unit time when the call is in progress. By Little's Theorem, these two models are stochastically equivalent if $R_k = R'_k/\mu_k$.

Since $g = \sum_{k=1}^{K} \lambda_k (1 - B_k) R_k$, maximizing the system revenue rate is equivalent to minimizing the weighted summation of the call-blocking probabilities. If we let $R_k = b_k/\mu_k$, then maximizing the system revenue rate is equivalent to maximizing the system throughput. Therefore, without loss of generality, we use the lump-sum revenue-collecting model, choose the system revenue rate g as the optimality criteria, and let $R_k = \gamma_k b_k/\mu_k$. γ_k is the *revenue coefficient* of class-k calls, which can be interpreted as the revenue generated by one unit of resource per unit time when the resource is used by a class-k call. Classes with higher revenue coefficients are more profitable.

Because the state and action sets are finite, and under the assumptions of Poisson call arrivals and bounded revenues, we only need to consider stationary Markov CAC policies in order to find the optimal CAC policy [4], [18]. In addition, in this paper, we only consider deterministic stationary CAC policies that generate a unichain [4], [18]. A discussion of multichain CAC policies is out of the scope of this paper.

Let $d(\mathbf{n}) = (a_1(\mathbf{n}), a_2(\mathbf{n}), \ldots, a_K(\mathbf{n}))$ be the decision rule when the system is in state $\mathbf{n}$. $a_k(\mathbf{n})$ takes value 0 or 1 if a class-k call is blocked or accepted, respectively, when the system is in state $\mathbf{n}$. A lump-sum revenue R_k is collected when a class-k call is accepted. Each policy d generates a controlled Markov reward process. Let Ω_d denote the state set, including all recurrent states, under d. For each state $\mathbf{n} \in \Omega_d$, let $\pi(\mathbf{n})$ be the steady-state probability that the system is in state $\mathbf{n}$. Set $\pi(\mathbf{n}) = 0$ for all state $\mathbf{n} \notin \Omega_d$. The detailed balance equations for the Markov process are [22]

$$\sum_{k=1}^{K}\{\pi(\mathbf{n}-e_k)\lambda_k a_k(\mathbf{n}-e_k) + \pi(\mathbf{n}+e_k)(n_k+1)\mu_k\}$$
$$= \pi(\mathbf{n})\sum_{k=1}^{K}\{\lambda_k a_k(\mathbf{n}) + n_k\mu_k\} \quad \text{for all } \mathbf{n} \in \Omega_d \tag{1}$$
$$\sum_{\mathbf{n}\in\Omega_d} \pi(\mathbf{n}) = 1. \tag{2}$$

e_k is the unit vector with the kth element being 1 and other elements being 0. Therefore, in general, a multidimensional birth–death process is required to model the system state under a CAC policy.

Once the equations are solved and the steady-state probabilities are known, performance metrics of interest, such as the call-blocking probabilities and the system revenue rate, can be directly evaluated. One brute-force approach to finding the optimal CAC policy is to search among all CAC policies, which is obviously not practical. A more efficient approach is to model the system as a Markov decision process (MDP), then MDP algorithms can be applied to compute the optimal CAC policy. [21] applied both the value iteration algorithm and the linear programming algorithm (see [4] or [18] for a full description of the MDP algorithms), and observed that the value iteration algorithm requires less CPU time.

We have implemented the value iteration algorithm. Using it, we can compute the optimal CAC policy for small K and C (e.g., $K = 2, C = 100$ or $K = 4, C = 20$). But for large C, and especially, large K, it faces the "curse of dimensionality" because the size of the state space grows exponentially with K and C. Hence, it is desirable to establish certain *structural properties* of the optimal CAC policy which enable efficient computation and easy implementation. Unfortunately, the optimal CAC policy, in general, does not have a simple structure (except for single-service systems) [3], [21]. Therefore, instead of finding the optimal CAC policy, one may want to find a suboptimal CAC policy among policies that have simple structures, if the structured policy is easy to compute and implement, and if its performance is close to that of the optimal CAC policy.

We now introduce some commonly employed structured CAC policies.

Definition 2.1: CS Policy: Under the CS policy, a call (of any class) is accepted upon arrival whenever there are available resources to handle the call, i.e., $a_k(\mathbf{n}) = 1$ if and only if $\mathbf{n} \cdot \mathbf{b}^T + b_k \le C$.

Definition 2.2: Complete Partitioning (CP) Policies: Under a CP policy, the system with capacity C is partitioned into K subsystems with capacities $C_1, \ldots, C_K$ such that $\sum_{k=1}^{K} C_k = C$. Subsystem k is dedicated to class-k calls, i.e., $a_k(\mathbf{n}) = 1$ if and only if $b_k(n_k + 1) \le C_k$.

Definition 2.3: Reservation Policies: Under a reservation policy, each class k is associated with a reservation parameter r_k. A class-k call is accepted if and only if the system has r_k or more free resources (reserved for other classes) after acceptance, i.e., $a_k(\mathbf{n}) = 1$ if and only if $\mathbf{n} \cdot \mathbf{b}^T + b_k \le C - r_k$. Note that the CS policy is a special reservation policy, in which $r_k = 0$ for all k.

Definition 2.4: Threshold Policies: Under a threshold policy, each class k is associated with a threshold parameter t_k. A class-k call is accepted if and only if there are available resources and the number of class-k calls in the system does not exceed t_k after acceptance, i.e., $a_k(\mathbf{n}) = 1$ if and only if $\mathbf{n} \cdot \mathbf{b}^T + b_k \le C$ and $n_k + 1 \le t_k$. Note that the threshold policies include the CS policy and all CP policies.

III. CAC for Single-Service Resource-Sharing Systems

A. *Structural Properties of the Optimal CAC Policy*

For single-service resource-sharing systems in which all classes have the same resource requirement and mean service time, i.e., $b_k = 1$ (without loss of generality) and $\mu_k = \mu$, [15] established the following two structural properties of the optimal CAC policy.

Property 3.1: If it is optimal to accept a class-k call when r units of resources are available, then it is also optimal to accept a class-k call when r' units of resources are available if $r' \ge r$.

Let $r_k = \min\{r$: the optimal CAC policy accepts a class-k call when r units of resources are available after acceptance$\}$. *Property 3.1* implies that the optimal CAC policy for single-service systems is a reservation policy with parameters $r_1, r_2, \ldots, r_K$.

Property 3.2: The optimal CAC policy always accepts calls from the class with the highest lump-sum revenue (i.e., the largest revenue coefficient, because $R_k = \gamma_k/\mu$) whenever there are available resources.

The following structural property of the optimal CAC policy was established in [3].

Property 3.3: If it is optimal to accept a class-k call when r units of resources are available, then it is also optimal to accept a class-k' call when r units of resources are available if $R_{k'} \ge R_k$ (i.e., $\gamma_{k'} \ge \gamma_k$).

We now show that these properties can be established from Bellman's optimality equation [4], [18] in a unified way. We first introduce a simplified system state description for single-service systems. Since all classes have the same resource requirement and mean service time, once a call is accepted, its class becomes irrelevant to the future evolution of the system. Therefore, we can use an integer $n \in \{0, 1, \ldots, C\}$ instead of a K-dimensional vector $\mathbf{n}$ to represent the system state, when there are n calls (of any class) in the system.

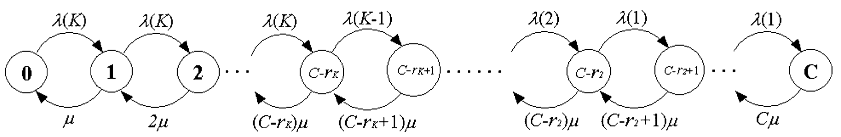


Fig. 1. 1-D birth–death process of the system state under a reservation policy.

We model the system as a semi-MDP (SMDP). The SMDP is observed at each call arrival and departure epoch. The state set of the SMDP is defined as

$$\mathcal{S} = \{s = (n,i) : n \in \{0,1,\ldots,C\}, i \in \{0,1,\ldots,K\}\}.$$

The SMDP is observed in state $(n,0)$ when a call departs, and subsequently, the system has n ongoing calls. In state $(n,0)$ the only action is to continue, so the action set $\mathcal{A}_{(n,0)} = \{0\}$. The state (n,i), for $1 \le i \le K$, is observed when the system has n ongoing calls and a class-i call arrives. In state (n,i), the system may accept or block the call, i.e., the action set $\mathcal{A}_{(n,i)} = \{1,0\}$.

Let $\beta(s,a)$ be the transition rate that the SMDP leaves state s when action $a \in \mathcal{A}_s$ is taken, so $1/\beta(s,a)$ is the expected length of time until the next decision epoch. We have

$$\beta((n,i),0) = \sum_{k=1}^{K} \lambda_k + n\mu \triangleq \beta_n, 0 \le i \le K$$
$$\beta((n,i),1) = \sum_{k=1}^{K} \lambda_k + (n+1)\mu \triangleq \beta_{n+1}, 1 \le i \le K.$$

Letting $q(s'|s,a)$ be the one-step state transition probability, we have

$$\begin{aligned} q((n,j)|(n,i),0) &= \lambda_j/\beta_n, 1 \le j \le K, 0 \le i \le K \\ q((n-1,0)|(n,i),0) &= n\mu/\beta_n, 0 \le i \le K \\ q((n+1,j)|(n,i),1) &= \lambda_j/\beta_{n+1}, 1 \le j \le K, 1 \le i \le K \\ q((n,0)|(n,i),1) &= (n+1)\mu/\beta_{n+1}, 1 \le i \le K \\ q(s'|s,a) &= 0, \text{ otherwise.} \end{aligned}$$

Letting $r(s,a)$ be the lump-sum revenue collected when the SMDP is in state s and action a is taken, we have $r((n,i),0) = 0$ for $0 \le i \le K$, and $r((n,i),1) = R_i = \gamma_i/\mu$ for $1 \le i \le K$.

The Bellman's optimality equation for the SMDP is

$$h(s) = \max_{a \in \mathcal{A}_s}\{r(s,a) - g^*/\beta(s,a) + \sum_{s' \in \mathcal{S}} q(s' \mid s,a)h(s')\} \quad (3)$$

where g^* is the revenue rate of the optimal CAC policy, and $h(s)$ is called the *bias* [18], or the *differential cost* [4] associated with state s. (3) can be further described as

$$h(n,0) = \frac{-g^*}{\beta_n} + \sum_{k=1}^{K} \frac{\lambda_k}{\beta_n} h(n,k) + \frac{n\mu}{\beta_n} h(n-1,0) \quad (4)$$
$$h(n,k) = \max\{h(n,0), R_k + h(n+1,0)\}, 1 \le k \le K. \quad (5)$$

Define $c(n) = h(n,0) - h(n+1,0)$ for $0 \le n \le C-1$ and $c(C) = \infty$. From (5), the optimal CAC policy is determined by $c(n)$ as follows:

$$a(n,k) = \begin{cases} 1, & \text{if } c(n) \le R_k \\ 0, & \text{otherwise.} \end{cases} \quad (6)$$

$c(n)$ can be viewed as the *system cost* of accepting a new call when the system has n ongoing calls. (6) has a nice interpretation that the optimal CAC policy accepts a call if and only if the collected revenue R_k exceeds or equals the system cost $c(n)$.

A direct consequence of (6) is that if it is optimal to accept a class-k call in state n, i.e., $c(n) \le R_k$, then it is also optimal to accept a class-k' call in state n, provided $R_{k'} \ge R_k$, because $c(n) \le R_k \le R_{k'}$. This establishes *Property 3.3*. In order to establish *Properties 3.1* and *3.2*, we need the following lemma.

Lemma 3.1: $c(n)$ is nonnegative and nondecreasing in all system states, i.e., $c(n) \ge 0$, $c(n) \le c(n+1)$.

Proof: See the Appendix.

From *Lemma 3.1*, if it is optimal to accept a class-k call in state n, i.e., $c(n) \le R_k$, then it is also optimal to accept a class-k call in state n', provided $n' \le n$, because $c(n') \le c(n) \le R_k$. This establishes *Property 3.1*. The proof of *Property 3.2* is given in the Appendix.

We order the call classes by their revenue coefficients so that $\gamma_1 \ge \gamma_2 \ge \cdots \ge \gamma_K$ (i.e., $R_1 \ge R_2 \ge \cdots \ge R_K$). Let $r_1^*, r_2^*, \ldots, r_K^*$ be the parameters of the optimal reservation policy. Combining *Properties 3.1, 3.2*, and *3.3*, we summarize the structural properties of the optimal CAC policy in the following theorem.

Theorem 3.1: For single-service systems, with $\gamma_1 \ge \gamma_2 \ge \cdots \ge \gamma_K$, the optimal CAC policy is a reservation policy with ordered parameters $r_1^*, r_2^*, \ldots, r_K^*$ such that $0 = r_1^* \le r_2^* \le \cdots \le r_K^* \le C$.

From *Theorem 3.1*, we only need to search among reservation policies with ordered parameters to find the optimal CAC policy. However, these theoretical results do not provide any practical method to determine the optimal reservation parameters.

B. Performance Evaluation of Reservation Policies

Under a reservation policy with parameters $0 = r_1 \le r_2 \le \cdots \le r_K \le C$, the system state can be modeled using a 1-D birth–death process, as shown in Fig. 1, where $\lambda(k) \triangleq \sum_{i=1}^{k} \lambda_i$. In any state n, the death rate (call departure rate) is $n\mu$. In state $0,1,\ldots,C-r_K-1$, the system accepts calls of all classes, so the birth rate is $\lambda(K)$. When the system enters state $C-r_K$, it does not accept class-K calls, so the birth rate is $\lambda(K-1)$. Birth rates in other states are similarly calculated.

Let $\rho_k = \lambda_k/\mu$ and $\bar{\rho}_k = \sum_{i=1}^{k} \rho_i$ for $1 \le k \le K$. Let r_{K+1} denote the system capacity. Let $\pi_C(n)$ be the steady-state probability that the system (with capacity C) is in state n. Based on the state-transition-rate diagram in Fig. 1, the steady-state probabilities of the birth–death process are shown in (7) and (8) at the bottom of the page.

$$\pi_C(0) = G(C)^{-1} = \left(1+\sum_{k=1}^{K}\sum_{n=C-r_{k+1}+1}^{C-r_k}\frac{\bar{\rho}_k^{n-C+r_{k+1}}\prod_{i=k+1}^{K}\bar{\rho}_i^{r_{i+1}-r_i}}{n!}\right)^{-1} \tag{7}$$

$$\pi_C(n) = \frac{1}{G(C)}\frac{\bar{\rho}_k^{n-C+r_{k+1}}\prod_{i=k+1}^{K}\bar{\rho}_i^{r_{i+1}-r_i}}{n!}, \quad \text{for } C-r_{k+1}+1\le n\le C-r_k, k=K,K-1,\ldots,1 \tag{8}$$

However, for large C, direct calculation faces a numerical overflow/underflow problem when calculating the *normalization constant* $G(C)$. We develop the following set of fast recursive formulas (for systems with arbitrary K and C) to solve this problem.

When a system with capacity C is operated under a reservation policy with parameters $0 = r_1 \le r_2 \le \cdots \le r_K \le C$, from (8), we have

$$\pi_C(C) = \frac{\bar{\rho}_1^{r_2}\prod_{i=2}^{K}\bar{\rho}_i^{r_{i+1}-r_i}}{G(C)C!}. \tag{9}$$

Similarly, for a system with capacity $C-1$ and reservation parameters $0 = r_1 \le r_2 - 1 \le \cdots \le r_K - 1 \le C-1$

$$\pi_{C-1}(C-1) = \frac{\bar{\rho}_1^{r_2-1}\prod_{i=2}^{K}\bar{\rho}_i^{r_{i+1}-r_i}}{G(C-1)(C-1)!}. \tag{10}$$

Comparing (9), (10), and from (7), we have

$$\pi_C(C) = \frac{\pi_{C-1}(C-1)}{\pi_{C-1}(C-1)+C/\bar{\rho}_1}. \tag{11}$$

(11) holds as long as $r_2 > 0$. When r_2 decreases to zero, the system accepts both class-1 and class-2 calls whenever there are available resources. Now the recursive equation becomes

$$\pi_n(n) = \frac{\pi_{n-1}(n-1)}{\pi_{n-1}(n-1)+n/\bar{\rho}_2}. \tag{12}$$

(12) holds as long as $r_3 > 0$. In general, for every n in $[C-r_{k+1}+1, C-r_k]$, $k = K, K-1, \ldots, 1$, we derive the following set of recursive formulas:

$$\pi_n(n) = \frac{\pi_{n-1}(n-1)}{\pi_{n-1}(n-1)+n/\bar{\rho}_k} \tag{13}$$

with $\pi_0(0) = 1$.

The following procedure can be applied to evaluate the performance metrics of a single-service system under any reservation policy.

Procedure 3.1. Performance Evaluation of Reservation Policies

```
1  $\pi(0) \leftarrow 1$
2  for $k \leftarrow K$ to $1$
3      for $n \leftarrow C - r_{k+1} + 1$ to $C - r_k$
4          $\pi(n) \leftarrow \frac{\pi(n-1)}{\pi(n-1)+n/\bar{\rho}_k}$
5  $\log G \leftarrow \sum_{i=1}^{K}(r_{i+1}-r_i)\log(\bar{\rho}_i) - \log(\pi(C)) - \log(C!)$
6  $\pi(0) \leftarrow \exp(-\log G)$
7  for $k \leftarrow K$ to $1$
8      for $n \leftarrow C - r_{k+1} + 1$ to $C - r_k$
9          $\log\pi(n) \leftarrow -\log G + (n - C + r_{k+1})\log(\bar{\rho}_k) + \sum_{i=k+1}^{K}(r_{i+1}-r_i)\log(\bar{\rho}_i) - \log(n!)$
10         $\pi(n) \leftarrow \exp(\log\pi(n))$
11 for $k \leftarrow 1$ to $K$
12     $B_k \leftarrow \sum_{n=C-r_k}^{C}\pi(n)$
13 $g \leftarrow \sum_{i=1}^{K}\lambda_i(1-B_i)R_i$
```

Steps 1–4 compute $\pi_C(C)$ recursively using (13). The recursive computation greatly reduces the numerical overflow/underflow problem. Steps 5–10 compute the normalization constant [based on (9)] and the steady-state probabilities [based on (7) and (8)]. Note that the logarithmic transformation converts the product into a sum and further reduces the numerical overflow/underflow problem. Steps 11-13 compute the call-blocking probabilities and the system revenue rate. The computational complexity of **Procedure 3.1** is $O(C)$.

Note that $r_1 = 0$ assures that the birth–death process in Fig. 1 is ergodic. Hence, it is time-reversible [23] and has product-form stationary distribution (7)–(8). The *insensitivity property* [5], [10], [22] guarantees that (7)–(8) still hold if the exponential service time distributions are replaced by arbitrary distributions with finite means. Exponential service times are good assumptions for many applications, but not for all. For example, for video streaming and live streaming applications, the session times are more or less deterministic. The insensitivity property enables us to extend the results here to arbitrary service time distributions.

C. Iterative Coordinate Search Algorithm for Reservation Policies

Given other parameters C, K, μ, and $(\gamma_k, \lambda_k)_{k=1,\ldots,K}$, the system revenue rate g can be viewed as a function of the reservation vector $\mathbf{r} \triangleq (r_1, r_2, \ldots, r_K) \in \mathcal{R} \triangleq \{0, 1, \ldots, C\}^K$, and can be evaluated using **Procedure 3.1**. Since $|\mathcal{R}| = (C+1)^K$, there exist $O(C^K)$ different reservation policies. Finding the optimal reservation policy using brute-force search becomes intractable for large C, and especially, large K. We propose an *iterative coordinate search algorithm* to find an *ordered coordinate optimal reservation policy* among all reservation policies.

Definition 3.1: A reservation vector $\mathbf{r} = (r_1, r_2, \ldots, r_K) \in \mathcal{R}$ is an ordered coordinate optimal point if it satisfies the following two properties:

$$r_0 \triangleq 0 = r_1 \le r_2 \le \cdots \le r_K \le r_{K+1} \triangleq C \tag{14}$$
$$g(r_1, \ldots, r_k, \ldots, r_K) \ge g(r_1, \ldots, \xi, \ldots, r_K) \tag{15}$$

for all ξ in $[r_{k-1}, r_{k+1}], k = 1, \ldots, K$. A reservation policy with reservation vector $\mathbf{r}$ is an ordered coordinate optimal reservation policy if $\mathbf{r}$ is an ordered coordinate optimal point.

Theorem 3.2: Assume that the call arrival rates are finite and the revenues are bounded. Then, for single-service systems, there exists at least one ordered coordinate optimal point in $\mathcal{R}$.

Proof: Under the assumption, the optimal CAC policy exists with a finite revenue rate. By *Theorem 3.1*, for single-service systems, the optimal CAC policy is a reservation policy with ordered parameters. Clearly, the parameters satisfy both (14) and (15), and thus constitute an ordered coordinate optimal point. □

Algorithm ICSA_RSV: Iterative Coordinate Search Algorithm for Reservation Policies

```
1. Choose an initial reservation vector (IRV) $\mathbf{r}^0$ that satisfies (14).
   Set $m = 1$.
2. Set $r_1^m = 0$, $r_{K+1}^m = C$.
   For $k = K, K-1, \ldots, 2$, search in $[r_{k-1}^{m-1}, r_{k+1}^m]$ to find $r_k^m$
   that maximizes the system revenue rate $g$:
   $r_k^m = \mathrm{argmax}_{\xi \in [r_{k-1}^{m-1}, r_{k+1}^m]} g(r_1^{m-1}, \ldots, r_{k-1}^{m-1}, \xi, r_{k+1}^m, \ldots, r_K^m)$.   (16)
   (In case more than one $\xi \in [r_{k-1}^{m-1}, r_{k+1}^m]$ achieves the same
   maximum $g$, the smallest $\xi$ is chosen to break the tie.)
3. If $\mathbf{r}^m \triangleq (r_1^m, r_2^m, \ldots, r_K^m) = \mathbf{r}^{m-1}$, stop.
   Otherwise, increase $m$ by 1 and go to step 2.
```

Theorem 3.3: Assume that the call arrival rates are finite and the revenues are bounded, ICSA_RSV converges in a finite number of iterations, and the returned reservation policy is an ordered coordinate optimal reservation policy.

Proof: Define $\mathbf{x}_k^m = (0, r_2^{m-1}, \ldots, r_k^{m-1}, r_{k+1}^m, \ldots, r_K^m)$ for $K \ge k \ge 1$. Note that $\mathbf{x}_K^m = \mathbf{r}^{m-1}$ and $\mathbf{x}_1^m = \mathbf{r}^m$. In the mth iteration, from (16), we have

$$g(\mathbf{r}^{m-1}) = g\left(\mathbf{x}_K^m\right) \le g\left(\mathbf{x}_{K-1}^m\right) \le \cdots \le g\left(\mathbf{x}_1^m\right) = g(\mathbf{r}^m) \tag{17}$$

i.e., g is increasing (nondecreasing) throughout the algorithm.

For any $\mathbf{r}$ that satisfies (14), define

$$l_k(\mathbf{r}) = \{(r_1, \ldots, r_{k-1}, \xi, r_{k+1}, \ldots, r_K) : r_{k-1} \le \xi \le r_{k+1}\}$$

which represents a *line segment* along the kth coordinate of $\mathcal{R}$. Let $\mathbf{L}_k = \{l_k(\mathbf{r}) : \mathbf{r} \text{ satisfies } (14)\}$ be the set including all (ordered) line segments along the kth coordinate of $\mathcal{R}$.

In the m-th iteration, ICSA_RSV searches along exactly one line segment $l_k(\mathbf{x}_k^m)$ in $\mathbf{L}_k$ for $K \ge k \ge 2$ in sequence. By the definition of $\mathbf{x}_{k-1}^m$, $l_k(\mathbf{r})$, and from (16), we have

$$\mathbf{x}_{k-1}^m = \mathrm{argmax}_{\mathbf{r} \in l_k(\mathbf{x}_k^m)} g(\mathbf{r}). \tag{18}$$

Assume, towards a contradiction, that ICSA_RSV searches the same line segment in some $\mathbf{L}_k$ during two different iterations $i < j$ without stopping, i.e., assume that ICSA_RSV does not converge after iteration j, but there exist some k and $i < j$ such that $l_k(\mathbf{x}_k^i) = l_k(\mathbf{x}_k^j)$.

Since r_k^m in (16) is uniquely determined, so is $\mathbf{x}_{k-1}^m$ in (18). Hence, $l_k(\mathbf{x}_k^i) = l_k(\mathbf{x}_k^j) \Rightarrow \mathbf{x}_{k-1}^i = \mathbf{x}_{k-1}^j \Rightarrow l_{k-1}(\mathbf{x}_{k-1}^i) = l_{k-1}(\mathbf{x}_{k-1}^j)$. Continuing induction, and finally, we have $\mathbf{x}_1^i = \mathbf{x}_1^j \Rightarrow \mathbf{r}^i = \mathbf{r}^j \Rightarrow g(\mathbf{r}^i) = g(\mathbf{r}^j)$. From (17), we have $g(\mathbf{r}^i) = g(\mathbf{r}^{i+1}) = \cdots = g(\mathbf{r}^j)$.

For any $\mathbf{r}$ and $\mathbf{r}'$ in $\mathcal{R}$, we say $\mathbf{r} \ge \mathbf{r}'$ if $r_k \ge r_k'$ for all k. If $g(\mathbf{r}^i) = g(\mathbf{r}^{i+1})$, then from (17)

$$g(\mathbf{r}^i) = g\left(\mathbf{x}_K^{i+1}\right) = g\left(\mathbf{x}_{K-1}^{i+1}\right) = \cdots = g\left(\mathbf{x}_1^{i+1}\right) = g(\mathbf{r}^{i+1}).$$

It follows that $\mathbf{r}^i = \mathbf{x}_K^{i+1} \ge \mathbf{x}_{K-1}^{i+1} \ge \cdots \ge \mathbf{x}_1^{i+1} = \mathbf{r}^{i+1}$, since the algorithm always chooses the smallest ξ in (16) to break the tie. Hence, we have $\mathbf{r}^i \ge \mathbf{r}^{i+1} \ge \cdots \ge \mathbf{r}^j$, and then from $\mathbf{r}^i = \mathbf{r}^j$, we have $\mathbf{r}^i = \mathbf{r}^{i+1}$. By the convergence criterion, the algorithm should converge at the $(i+1)$th iteration, a contradiction.

Therefore, during every iteration, ICSA_RSV searches a distinct line segment in $\mathbf{L}_k$ for $K \ge k \ge 2$ before it converges. Since $|\mathbf{L}_k| < (C+1)^{K-2}$, we conclude that ICSA_RSV converges in a finite number of iterations that is bounded by $(C+1)^{K-2}$.

When ICSA_RSV converges after iteration M, i.e., $\mathbf{r}^M = \mathbf{r}^{M-1}$, then from (16), it is clear that $\mathbf{r}^M$ is an ordered coordinate optimal point and the returned policy is an ordered coordinate optimal reservation policy. □

For $K = 2$, it is clear that ICSA_RSV always returns the optimal reservation policy in two iterations. In general, since ICSA_RSV is a *local search algorithm* [1], two important issues need to be investigated: 1) the convergence rate of the algorithm;

TABLE I
NUMERICAL EXPERIMENTS FOR SINGLE-SERVICE RESOURCE-SHARING SYSTEMS: $K = 4, C = 20$

$(\gamma_1, \gamma_2, \gamma_3, \gamma_4) = (8, 4, 2, 1)$, $b_k = 1, \mu_k = 1$						
η	Traffic Distribution	CS	Reservation Policy		Optimal CAC Policy	
			ICSA_RSV (IRV1)		Value Iteration	
		g	$\mathbf{r}$	$g(M_r)$	$\mathbf{r}^*$	g^*
1.0	UNIFORM	100.0	(0,0,2,5)	107.5(3)	(0,0,2,5)	107.5
	HIGH_HIGH	100.0	(0,1,4,9)	106.4(3)	(0,1,4,9)	106.4
	HIGH_LOW	100.0	(0,0,1,3)	106.2(2)	(0,0,1,3)	106.2
	RANDOM	100.0	(0,0,2,6)	109.5(2)	(0,0,2,6)	109.5
1.6	UNIFORM	100.0	(0,1,5,16)	130.1(3)	(0,1,5,16)	130.1
	HIGH_HIGH	100.0	(0,2,11,20)	122.3(3)	(0,2,11,20)	122.3
	HIGH_LOW	100.0	(0,0,2,8)	127.9(3)	(0,0,2,8)	127.9
	RANDOM	100.0	(0,1,4,11)	129.2(3)	(0,1,4,11)	129.2

2) the quality of the returned reservation policy. One would also expect that the IRV $\mathbf{r}^0$ we choose may affect the performance of ICSA_RSV. These will be investigated in the next subsection.

D. Numerical Experiments

We have conducted extensive numerical experiments to evaluate the performance of ICSA_RSV for single-service resource-sharing systems. Part of the results are shown in Tables I and II. As a comparison, for all cases we evaluate the performance of the CS policy. We scale the revenue rate of the CS policy to be 100.0. The revenue rates of other policies are also scaled according to the CS policy, so that percentage improvements of other policies over the CS policy are clearly shown.

Let $\eta_k = (\lambda_k b_k / \mu_k)/C$ be the (normalized) traffic demand of class-k calls, and $\eta = \sum_{k=1}^{K} \eta_k$ be the (normalized) total traffic demand of all classes. For any given η, we consider the following four traffic distributions among different classes.

1) UNIFORM: $\eta_k = \overline{\eta} \triangleq \eta/K$ for all k.
2) HIGH_HIGH: $\eta_k = (3/2)\overline{\eta}$ for $1 \le k \le (K/2)$, $\eta_k = (1/2)\overline{\eta}$ for $(K/2)+1 \le k \le K$, i.e., higher-profit classes have higher traffic demands.
3) HIGH_LOW: $\eta_k = (1/2)\overline{\eta}$ for $1 \le k \le (K/2)$, $\eta_k = (3/2)\overline{\eta}$ for $(K/2)+1 \le k \le K$, i.e., higher-profit classes have lower traffic demands.
4) RANDOM: η_k is a random number uniformly chosen in $[(1/2)\overline{\eta}, (3/2)\overline{\eta}]$ for all k.

For each set of K and C, we have done experiments over a wide range of η. In general, higher η and/or larger differences among the revenue coefficients result in greater improvement of the returned reservation policy over the CS policy and other heuristic reservation policies (HRPs). Due to space constraints, we only list the results for $\eta = 1.0$ (critically loaded) and $\eta = 1.6$ (overloaded). The revenue coefficients are within the same order of magnitude.

The IRV $\mathbf{r}^0$ of ICSA_RSV is chosen as follows.

1) IRV1: $r_k^0 = 0$ for all k, this is the CS policy.
2) IRV2: $r_1^0 = 0, r_k^0 = C$ for $k > 1$.
3) IRV3: $r_1^0 = 0, r_k^0$ is chosen at random uniformly in $[0, r_{k+1}]$, for $k = K, \ldots, 2$ $(r_{K+1}^0 = C)$.

1) Convergence Rate and Computational Complexity: Let M_r be the number of iterations required by ICSA_RSV to converge. We found that for all cases, ICSA_RSV converges very fast, with M_r on the order of K. For small K $(K = 4)$, we found that M_r and the returned reservation policy are insensitive to IRV1–IRV3, so in Table I, for each case we only show one set of results $(\mathbf{r}, g, M_r)$ of ICSA_RSV with IRV1.

For large $K(K = 16)$, we found that ICSA_RSV converges faster when starting from IRV1 or IRV3 than starting from IRV2. The returned reservation policies may be different with different IRVs, but they have the same revenue rate, as shown in Table II.

Let T_r be the computational complexity of evaluating the performance of a reservation policy. In our experiments $T_r = O(C)$ due to **Procedure 3.1**. During each iteration, ICSA_RSV searches at most $(C+1)(K-1)$ different reservation policies. In practice, the number of iterations M_r is on the order of K. Hence, the computational complexity of ICSA_RSV is $O(M_r C K T_r) = O(C^2 K^2)$ in practice. It can be applied for systems with large C and K. For example, for $K = 16$ and $C = 200$, on average ICSA_RSV converges in a few seconds, running on a PC with Pentium IV 1.5 GHz CPU.

2) Quality of the Returned Reservation Policy: For small K and C, e.g., $K = 4, C = 20$ (Table I), we can compute the optimal CAC policy using the value iteration algorithm. For single-service systems, the optimal CAC policy coincides with the optimal reservation policy. It is rather striking that for all of the cases we studied, the ordered coordinate optimal reservation policy returned by ICSA_RSV is exactly the optimal CAC policy!

For large K and C, e.g., $K = 16$ and $C = 200$ (Table II), it is infeasible to find the optimal reservation policy using the value iteration algorithm or brute-force search. We compare the performance of the returned reservation policy with two HRPs. We first determine the number $\widetilde{K}$ such that $\sum_{k=1}^{\widetilde{K}} \lambda_k/\mu \le C < \sum_{k=1}^{\widetilde{K}+1} \lambda_k/\mu$.

HRP1 sets its reservation parameters as follows:

$$r_k = 0 \text{ for } 1 \le k \le \widetilde{K}, \; r_k = 5\%C \text{ for } \widetilde{K}+1 \le k \le K.$$

HRP2 sets its reservation parameters as follows:

$$r_k = 0 \text{ for } 1 \le k \le \widetilde{K}, \; r_k = 10\%C \text{ for } \widetilde{K}+1 \le k \le K.$$

We found that for all cases, the reservation policy returned by ICSA_RSV outperforms the two HRPs. Indeed, if we choose the IRV r^0 based on a good HRP, then ICSA_RSV will return a reservation policy that performs better than, or at least as well as, the HRP, because g is increasing throughout the algorithm.

IV. CAC FOR MULTISERVICE RESOURCE-SHARING SYSTEMS

For multiservice resource-sharing systems, calls of different classes may have different resource requirements and mean service times. Unlike single-service systems, the optimal CAC

TABLE II
NUMERICAL EXPERIMENTS FOR SINGLE-SERVICE RESOURCE-SHARING SYSTEMS: $K = 16, C = 200$

$\gamma = (48, 44, 40, 36, 32, 28, 24, 20, 16, 8, 4, 2, 1, 0.5, 0.25, 0.125)$, $b_k = 1, \mu_k = 1$							
η	Traffic Distribution	CS	Reservation Policy				
			ICSA_RSV			Heuristic	
			IRV1	IRV2	IRV3	HRP1: 5%	HRP2: 10%
		g	$g(M_r)$	$g(M_r)$	$g(M_r)$	g	g
1.0	UNIFORM	100.0	105.5(4)	105.5(20)	105.5(4)	100.0	100.0
	HIGH_HIGH	100.0	105.3(6)	105.3(18)	105.3(6)	100.0	100.0
	HIGH_LOW	100.0	105.5(4)	105.5(22)	105.5(4)	100.0	100.0
	RANDOM	100.0	111.6(5)	111.6(22)	111.6(5)	107.9	109.6
1.6	UNIFORM	100.0	152.4(5)	152.4(19)*[1]	152.4(5)	143.7	147.6
	HIGH_HIGH	100.0	133.5(5)	133.5(16)*	133.5(5)	128.8	129.9
	HIGH_LOW	100.0	156.8(6)	156.8(24)*	156.8(6)	152.1	155.3
	RANDOM	100.0	147.6(6)	147.6(19)*	147.6(6)	141.8	145.0

1. '*' means that the returned reservation policy with this initial reservation vector differs from the returned reservation policy with IRV1.

policy does not have a simple structure. Again, it is prohibitively difficult to compute the optimal CAC policy using any MDP algorithm because of the "curse of dimensionality."

Reservation policies can be implemented for multiservice systems as well. However, a multidimensional birth–death process is required to model the system state under a reservation policy. In order to evaluate the system performance, we need to solve the detailed balance equations (1)-(2), which requires excessive computational effort. This makes the task of finding the optimal reservation policy extremely challenging.

Therefore, we are motivated to consider the *threshold* policies, a family of structured CAC policies under which the stationary distribution of the system state has a nice product form.

A. Performance Evaluation of Threshold Policies

The threshold policies form a subset of the *coordinate convex* policies [7], [20], [22]. A CAC policy with the associated admissible state set Ω is said to be *coordinate convex* if:

1) $\Omega \subseteq \Omega_{\text{cs}}$ is a nonempty set that satisfies: $\forall \mathbf{n} \in \Omega$, if $n_k > 0$, then $\mathbf{n} - e_k \in \Omega$;
2) a class-k call is accepted if and only if the system state remains in Ω after acceptance, i.e., $a_k(\mathbf{n}) = 1$ if and only if $\mathbf{n} + e_k \in \Omega$.

It is clear that a threshold policy with parameters $t_1, t_2, \ldots, t_K$ is coordinate convex, with the associated admissible state set

$$\Omega_t = \{\mathbf{n} : \mathbf{n} \cdot \mathbf{b}^T \leq C, \quad 0 \leq n_k \leq t_k \text{for} 1 \leq k \leq K\}.$$

Under a coordinate convex policy, the induced Markov process is time-reversible and its stationary distribution has a product form, as summarized in the following theorem [20], [22].

Theorem 4.1: If the system is operated under a coordinate convex CAC policy with the associated admissible state set Ω, then the induced Markov process is time-reversible and its steady-state probabilities are given by

$$\pi(\mathbf{n}) = \frac{1}{G} \prod_{k=1}^{K} \frac{\rho_k^{n_k}}{n_k!}, \mathbf{n} \in \Omega \tag{19}$$

$$\text{where } G \triangleq \sum_{\mathbf{n} \in \Omega} \prod_{k=1}^{K} \frac{\rho_k^{n_k}}{n_k!}, \rho_k \triangleq \frac{\lambda_k}{\mu_k}. \tag{20}$$

The insensitivity property for product-form loss systems/networks enables us to extend the results of this section to arbitrary service time distributions [5], [10], [22]. Direct calculation of (19)–(20) is not efficient, since it requires enumerating all states in Ω. It also faces the numerical overflow/underflow problem for large C when calculating the normalization constant G. The convolution algorithm with binary tree implementation proposed in [24] can be applied to evaluate the system performance under any threshold policy, with a computational complexity of $O(C^2 K \log K)$.

Threshold policies are a rich family of structured CAC policies that include the CS policy and all CP policies. When $K = 2$, it was shown in [20] that for a wide range of parameters, the optimal threshold policy is optimal over all coordinate convex policies. We are interested in developing fast and efficient search algorithms to find the optimal or near-optimal threshold policies.

B. Iterative Coordinate Search Algorithm for Threshold Policies

Given other parameters $C, K, (\gamma_k, b_k, \lambda_k, \mu_k)_{k=1,\ldots,K}$, the system revenue rate g can be viewed as a function of the threshold vector $\mathbf{t} \triangleq (t_1, t_2, \ldots, t_K) \in \mathcal{T} \triangleq \mathcal{T}_1 \times \mathcal{T}_2 \times \cdots \times \mathcal{T}_K$, where $\mathcal{T}_k \triangleq \{0, 1, \ldots, N_k \triangleq \lfloor \frac{C}{b_k} \rfloor\}$. Since $|\mathcal{T}| = \prod_{k=1}^{K} |\mathcal{T}_k| = \prod_{k=1}^{K} (N_k + 1)$, there exist $\prod_{k=1}^{K} (N_k + 1)$, or $O(C^K)$ different threshold policies. Finding the optimal threshold policy using

TABLE III
NUMERICAL EXPERIMENTS FOR MULTISERVICE RESOURCE-SHARING SYSTEMS: $K = 4, C = 20$

$(\gamma_1, \gamma_2, \gamma_3, \gamma_4) = (8, 4, 2, 1)$, Traffic Distribution: RANDOM								
Parameters			CS	Threshold Policy				Optimal CAC Policy
				ICSA_THD (ITV1)		Brute-Force Search		Value Iteration
η	$(b_1, ..., b_4)$	$(\mu_1, ..., \mu_4)$	g	$(t_1b_1, ..., t_4b_4)^1$	$g(M_t)$	$(t_1b_1, ..., t_4b_4)$	g	g^*
1.0	(5,4,2,1)	(5,4,2,1)	100.0	(20,20,6,0)	115.1(2)	(20,20,6,0)	115.1	116.2
		(1,1,1,1)	100.0	(20,20,10,0)	112.0(2)	(20,20,10,0)	112.0	114.2
		(1,2,4,5)	100.0	(20,20,6,0)	116.7(2)	(20,20,6,0)	116.7	121.2
	(1,2,4,5)	(5,4,2,1)	100.0	(20,20,12,0)	103.4(2)	(20,20,12,0)	103.4	103.4
		(1,1,1,1)	100.0	(20,20,8,0)	103.3(2)	(20,20,8,0)	103.3	103.8
		(1,2,4,5)	100.0	(20,20,8,0)	104.6(2)	(20,20,8,0)	104.6	106.2
1.6	(5,4,2,1)	(5,4,2,1)	100.0	(20,8,0,0)	160.2(2)	(20,8,0,0)	160.2	161.5
		(1,1,1,1)	100.0	(20,8,0,0)	160.7(2)	(20,8,0,0)	160.7	163.9
		(1,2,4,5)	100.0	(20,4,0,0)	181.8(2)	(20,4,0,0)	181.8	192.5
	(1,2,4,5)	(5,4,2,1)	100.0	(20,20,12,0)	105.4(2)	(20,20,12,0)	105.4	105.5
		(1,1,1,1)	100.0	(20,20,8,0)	108.1(2)	(20,20,8,0)	108.1	109.4
		(1,2,4,5)	100.0	(20,20,0,0)	107.1(2)	(20,20,0,0)	107.1	108.6

1. We show t_kb_k for each class in order to compare the threshold effects for classes with different resource requirements.

a brute-force search becomes intractable for large C, and especially, large K.

With minor modifications, the iterative coordinate search algorithm proposed for reservation policies can be applied to find a *coordinate optimal threshold policy* among all threshold policies.

Definition 4.1: A threshold vector $\mathbf{t} = (t_1, t_2, \ldots, t_K) \in \mathcal{T}$ is a coordinate optimal point if it satisfies the following coordinate optimal property:

$$g(t_1, \ldots, t_k, \ldots, t_K) \geq g(t_1, \ldots, \xi, \ldots, t_K) \qquad (21)$$

for all $\xi \in \mathcal{T}_k = \{0, 1, \ldots, N_k\}, k = 1, \ldots, K$. A threshold policy with threshold vector $\mathbf{t}$ is a coordinate optimal threshold policy if $\mathbf{t}$ is a coordinate optimal point.

Note that if we model the problem of finding the optimal threshold policy as a K-player cooperative game, with each class k choosing its own threshold parameter t_k, and all classes have the same payoff function $g(t_1, t_2, \ldots, t_K)$, then a coordinate optimal point that satisfies (21) is a Nash equilibrium of the cooperative game.

Theorem 4.2: Assume that the call arrival rates are finite and the revenues are bounded, then there exists at least one coordinate optimal point in $\mathcal{T}$.

Proof: The parameters of the optimal threshold policy constitute an ordered coordinate optimal point in $\mathcal{T}$. □

We order the classes by their revenue coefficients so that $\gamma_1 \geq \gamma_2 \geq \cdots \geq \gamma_K$. If $\gamma_k = \gamma_{k+1}$, then class k and $k+1$ are ordered according to b, such that $b_k \geq b_{k+1}$.

Algorithm ICSA_THD Iterative Coordinate Search Algorithm for Threshold Policies

1. Choose an initial threshold vector (ITV) $\mathbf{t}^0$. Set $m = 1$.
2. For $k = 1, 2, \ldots, K$, search along the kth coordinate set $\mathcal{T}_k$ in sequence to find t_k^m such that

$$t_k^m = \arg\max_{\xi \in \mathcal{T}_k} g(t_1^m, \ldots, t_{k-1}^m, \xi, t_{k+1}^{m-1}, \ldots, t_K^{m-1}). \qquad (22)$$

   (In case more than one $\xi \in \mathcal{T}_k$ achieves the same maximum g, the largest ξ is chosen to break the tie.)
3. If $\mathbf{t}^m \triangleq (t_1^m, t_2^m, \ldots, t_K^m) = \mathbf{t}^{m-1}$, stop. Otherwise, increase m by 1 and go to step 2.

Theorem 4.3: Assume that the call arrival rates are finite and the revenues are bounded, ICSA_THD converges in a finite number of iterations, and the returned threshold policy is a coordinate optimal threshold policy.

Proof: The proof is similar to that of *Theorem 3.3*. □

C. Efficiency versus Fairness

We observed that the threshold policy returned by ICSA_THD may exclusively block some low-profit classes to maximize the system revenue rate, i.e., $t_k = 0$ for some class k. In order to prevent such unfairness, we can change the $\mathcal{T}_k$ in *Definition 4.1* and Algorithm ICSA_THD to be the set $\{\bar{t}_k, \bar{t}_k + 1, \ldots, N_k\}$ for some $\bar{t}_k > 0$. Clearly, the statements of *Theorems 4.2* and *4.3* still hold. In addition, the returned

TABLE IV
NUMERICAL EXPERIMENTS FOR MULTISERVICE RESOURCE-SHARING SYSTEMS: $K = 16, C = 200$

$(\gamma_1, ..., \gamma_{16}) = (48, 44, 40, 36, 32, 28, 24, 20, 16, 8, 4, 2, 1, 0.5, 0.25, 0.125)$, Traffic Distribution: RANDOM								
Parameters			CS	Threshold Policy				
				ICSA_THD				RCS
				ITV1	ITV2	ITV3	ITV4	Heuristics
η	$(b_1, ..., b_{16})$	$(\mu_1, ..., \mu_{16})$	g	$g(M_t)$	$g(M_t)$	$g(M_t)$	$g(M_t)$	g
1.0	(20,20,15,15,10,10, 8,8,5,5,4,4,2,2,1,1)	(20,20,15,15,10,10,8,8,5,5,4,4,2,2,1,1)	100.0	130.1(4)	130.1(5)	130.1(4)	130.1(4)	109.1
		(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)	100.0	127.9(3)	127.9(4)	127.9(3)	127.9(4)	106.6
		(1,1,2,2,4,4,5,5,8,8,10,10,15,15,20,20)	100.0	113.0(4)	113.0(5)	113.0(4)	113.0(5)	100.0
	(1,1,2,2,4,4,5,5,8,8, 10,10,15,15,20,20)	(20,20,15,15,10,10,8,8,5,5,4,4,2,2,1,1)	100.0	107.2(4)	107.2(5)*^{1}	107.2(4)	107.2(5)	103.0
		(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)	100.0	107.2(5)	107.2(5)*	107.2(5)	107.2(6)*	103.1
		(1,1,2,2,4,4,5,5,8,8,10,10,15,15,20,20)	100.0	103.4(5)	103.4(7)	103.4(5)	103.4(7)	100.0
1.6	(20,20,15,15,10,10, 8,8,5,5,4,4,2,2,1,1)	(20,20,15,15,10,10,8,8,5,5,4,4,2,2,1,1)	100.0	220.5(4)	220.5(5)	220.5(4)	220.5(5)	214.1
		(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)	100.0	214.8(4)	214.8(5)	214.8(4)	214.8(5)*	202.6
		(1,1,2,2,4,4,5,5,8,8,10,10,15,15,20,20)	100.0	220.3(3)	220.3(4)	220.3(3)	220.3(4)	210.1
	(1,1,2,2,4,4,5,5,8,8, 10,10,15,15,20,20)	(20,20,15,15,10,10,8,8,5,5,4,4,2,2,1,1)	100.0	113.7(3)	113.7(4)	113.7(3)	113.7(4)	112.4
		(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)	100.0	116.3(4)	116.3(4)	116.3(4)	116.3(4)*	115.3
		(1,1,2,2,4,4,5,5,8,8,10,10,15,15,20,20)	100.0	113.2(4)	113.2(4)*	113.2(4)	113.2(4)*	112.6

1. '*' means that the returned threshold policy with this initial threshold vector differs from the returned threshold policy with ITV1.

threshold policy satisfies that $t_k \geq \bar{t}_k$, i.e., every class has at least some share of the resources.

In general, there is a tradeoff between *efficiency* (achieving high system revenue rate) and *fairness* (providing fair share of resources to different classes) for CAC policies. A game-theoretic framework on fair–efficient CAC policies was presented in [6].

D. Numerical Experiments

We have conducted extensive numerical experiments to evaluate the performance of ICSA_THD. Part of the results are shown in Tables III and IV.

The ITV $\mathbf{t}^0$ of ICSA_THD is chosen as follows.

1) ITV1: $t_k^0 = 0$ for all k.
2) ITV2: $t_k^0 = N_k$ for all k, this is the CS policy.
3) ITV3: t_k^0 is chosen at random uniformly in $[0, N_k]$ for all k.
4) ITV4: $\mathbf{t}^0$ is chosen according to the heuristics proposed in [8].

1) Convergence Rate and Computational Complexity: Let M_t be the number of iterations required by ICSA_THD to converge. We found that for all cases, ICSA_THD converges very fast, with M_t on the order of K. For small K ($K = 4$), we found that M_t and the returned threshold policy are insensitive to ITV1 to ITV4. Therefore, in Table III, for each case we only list one set of results $(\mathbf{t}, g, M_t)$ of ICSA_THD with ITV1.

For large $K (K = 16)$, we found that ICSA_THD converges slightly faster when starting from ITV1 or ITV3 than starting from ITV2 or ITV4. The returned threshold policies may be different with different ITVs, but they have the same revenue rate.

Let T_t be the computational complexity of evaluating the performance of a threshold policy. In our experiments, $T_t = O(C^2 K \log K)$ due to the convolution algorithm with binary tree implementation [24]. During each iteration, ICSA_THD searches $\sum_{k=1}^{K}(N_k + 1)$, or $O(CK)$ different threshold policies. In practice, M_t is on the order of K. Hence, the computational complexity of ICSA_THD is $O(M_t C K T_t) = O(C^3 K^3 \log K)$, in practice. It can be applied for systems with large C and K. For example, for $K = 16$ and $C = 200$, on average, ICSA_THD converges in 1-2 min, running on a PC with Pentium IV 1.5 GHz CPU.

2) Quality of the Returned Threshold Policy: For small K and C, e.g., $K = 4$ and $C = 20$ (Table III), we are able to find the optimal threshold policy using brute-force search. It is rather striking that for all of the cases we studied, the coordinate optimal threshold policy returned by ICSA_THD is exactly the optimal threshold policy! For these cases, we can also find the optimal CAC policy using the value iteration algorithm. We found that the revenue rate of the optimal threshold policy is very close to that of the optimal CAC policy.

For large K and C, e.g., $K = 16$ and $C = 200$ (Table IV), it is infeasible to find the optimal threshold policy using brute-force search. We compare the performance of the returned threshold policy with the *restricted CS* (RCS) policy (a specific threshold policy with parameters chosen heuristically) proposed in [8]. For all cases, the threshold policy returned by ICSA_THD outperforms the RCS policy. Indeed, if we choose the ITV $t0$ based on a good heuristic threshold policy, then ICSA_THD will return a threshold policy that performs better than, or at least as well as, the heuristic threshold policy, because g is increasing throughout the algorithm.

V. Comparison Between Reservation and Threshold Policies

Both reservation policies and threshold policies can be implemented for single-service/multiservice resource-sharing systems. Algorithm ICSA_RSV together with **Procedure 3.1** can be applied to find an ordered-coordinate optimal reservation policy for single-service systems. But for multiservice systems, a multidimensional birth–death process is required to model the system state under a reservation policy, which requires excessive computation effort to evaluate its performance.

In [16], we proposed a *uniformization technique* to convert a multiservice system into a single-service system. Then we can apply ICSA_RSV to find a reservation policy for the converted system, and implement it for the original system. Note that since the converted system and the original system are not stochastically equivalent, the returned reservation policy is not necessarily an ordered-coordinate optimal reservation policy for the original system. This may cause some performance loss [16].

On the other hand, Algorithm ICSA_THD, together with the convolution algorithms proposed in [24], can be applied to find a coordinate optimal threshold policy for both single-service and multiservice systems.

In [16], we compared the revenue rate of the reservation policy and the threshold policy returned by the coordinate search algorithms: for all single-service system experiments, the returned reservation policy always yields a higher revenue rate than the returned threshold policy; while for multiservice system experiments, the returned threshold policy has a higher revenue rate in most cases with a few exceptions. Therefore, for single-service systems, we suggest applying ICSA_RSV to find a reservation policy; for multiservice systems, we suggest applying ICSA_THD to find a threshold policy.

Other performance metrics associated with a CAC policy, such as *fairness* and *robustness* (sensitivity to changing traffic demands), may also affect our choice of the CAC policy to use. Through numerical experiments, it was shown in [6] and [16], respectively, that reservation policies are more fair and robust than threshold policies. Hence, it will be an important future work to develop an efficient procedure to evaluate the performance of reservation policies for multiservice systems, so that the iterative coordinate search algorithm proposed in this paper can be used to find a high-performance reservation policy for multiservice systems.

VI. Conclusion

In this paper, we consider CAC for both single-service and multiservice resource-sharing systems. For most systems of practical interest, computing the optimal CAC policy is prohibitively difficult because of the "curse of dimensionality." Therefore, we are motivated to consider two families of structured CAC policies: reservation and threshold policies. These policies are easy to implement and have good performance, in practice. However, it is difficult to find the optimal structured policies among all structured policies. In practice, the parameters of these structured CAC policies are chosen empirically or based on simple heuristics.

The contributions of this paper include the fast recursive formulas developed to evaluate the performance of any reservation policy for single-service systems, and the iterative coordinate search algorithms proposed to determine the parameters of the structured policies, with the objective of maximizing the system revenue rate. We prove the convergence of the search algorithms. Through extensive numerical experiments, we show that the search algorithms converge quickly and work for systems with large capacity and many call classes. In addition, the returned structured policies have optimal or near-optimal performance, and outperform those structured policies with parameters chosen based on simple heuristics.

For real systems, the call arrivals are not necessarily stationary, and the traffic demands may not be known beforehand. In practice, adaptive schemes (as in [25]) can be implemented to estimate the call arrival rates (traffic demands), and the system dynamically adjusts the parameters of the structured CAC policy using the search algorithms developed in this paper. The high efficiency of the search algorithms makes such online optimization possible, even for systems with large capacity and many call classes.

There are many extensions of this single system with single resource (SSSR) model. First, we can extend a single-loss system to a loss network [12]. For example, in telecommunication networks, instead of considering CAC for a single access link, we can consider CAC and routing for the whole network. Second, we can extend a single-resource model to a multiresource model. For example, in wireless networks, in addition to channels, power is also an important resource. A common approach to studying these extended models is to decompose a complex model into a collection of SSSR models. Under the assumption that these SSSR models are statistically independent, each with a reduced offered load, the search algorithms developed in this paper can be applied to each one of the SSSR models.

Appendix

Lemma 3.1: $c(n)$ is nonnegative and nondecreasing in all system states, i.e., $c(n) \geq 0$, $c(n) \leq c(n+1)$.

Proof: We first prove $c(n) \geq 0$. By Bellman's optimality (4) and (5)

$$\begin{aligned} h(n,0) &= \frac{-g^*}{\beta_n} + \sum_{k=1}^{K} \frac{\lambda_k}{\beta_n} h(n,k) + \frac{n\mu}{\beta_n} h(n-1,0) \\ &\geq \frac{-g^*}{\beta_n} + \sum_{k=1}^{K} \frac{\lambda_k}{\beta_n} (R_k + h(n+1,0)) + \frac{n\mu}{\beta_n} h(n-1,0) \end{aligned}$$

$$\beta_n = \sum_{k=1}^{K} \lambda_k + n\mu$$

$$\begin{aligned} &\Rightarrow (h(n,0) - h(n+1,0)) \sum_{k=1}^{K} \lambda_k \\ &\geq \sum_{k=1}^{K} \lambda_k R_k - g^* + n\mu((h(n-1,0) - h(n,0)). \end{aligned}$$

Letting $c(n) = h(n,0) - h(n+1,0)$ for $0 \le n \le C-1$ and $c(C) = \infty$, we have

$$c(n)\sum_{k=1}^{K}\lambda_k \ge \sum_{k=1}^{K}\lambda_k R_k - g^* + n\mu c(n-1). \tag{23}$$

When $n = 0$, since $g^* \le \sum_{k=1}^{K}\lambda_k R_k$

$$c(0)\sum_{k=1}^{K}\lambda_k \ge \sum_{k=1}^{K}\lambda_k R_k - g^* \ge 0.$$

Hence, $c(0) \ge 0$. Continuing induction on (23), we have $c(n) \ge 0$ for all system state n.

Now we prove the second part of *Lemma 3.1*. Combining (4) and (5), we have

$$\begin{aligned}
&\beta_n h(n,0)\\
&= -g^* + \sum_{k=1}^{K}\lambda_k h(n,k) + n\mu h(n-1,0)\\
&= -g^* + \sum_{k=1}^{K}\lambda_k \max\{h(n,0), R_k + h(n+1,0)\}\\
&\quad + n\mu h(n-1,0)\\
&\Rightarrow n\mu(h(n,0) - h(n-1,0))\\
&= -g^* + \sum_{k=1}^{K}\lambda_k(\max\{h(n,0), R_k + h(n+1,0)\}\\
&\quad - h(n,0))\\
&= -g^* + \sum_{k=1}^{K}\lambda_k \max\{0, R_k + h(n+1,0) - h(n,0)\}.
\end{aligned}$$

Equivalently

$$\begin{aligned}
n\mu c(n-1) &= g^* - \sum_{k=1}^{K}\lambda_k \max\{0, R_k - c(n)\}\\
&= g^* + \sum_{k=1}^{K}\lambda_k \min\{0, c(n) - R_k\}. \qquad (24)
\end{aligned}$$

When $n = 0$, (24) $\Rightarrow$

$$g^* + \sum_{k=1}^{K}\lambda_k \min\{0, c(0) - R_k\} = 0. \tag{25}$$

When $n > 0$, (24) $\Rightarrow$

$$c(n-1) = \frac{g^* + \sum_{k=1}^{K}\lambda_k \min\{0, c(n) - R_k\}}{n\mu} \tag{26}$$

$$c(n) = \frac{g^* + \sum_{k=1}^{K}\lambda_k \min\{0, c(n+1) - R_k\}}{(n+1)\mu}. \tag{27}$$

Assume, towards a contradiction, that $c(n) > c(n+1)$ for some n, then

$$\sum_{k=1}^{K}\lambda_k \min\{0, c(n) - R_k\} \ge \sum_{k=1}^{K}\lambda_k \min\{0, c(n+1) - R_k\}.$$

Since $n\mu < (n+1)\mu$ and both $c(n-1)$ and $c(n)$ are nonnegative, comparing (26) and (27), we have $c(n-1) \ge c(n)$, with equality if and only if $c(n-1) = c(n) = 0$, which implies $0 = c(n) > c(n+1)$, a contradiction to $c(n+1) \ge 0$. Therefore, $c(n-1) > c(n)$. Continuing induction, we have $c(0) > c(1)$. From (27) and (25)

$$\begin{aligned}
c(0) &= \frac{g^* + \sum_{k=1}^{K}\lambda_k \min\{0, c(1) - R_k\}}{\mu}\\
&\le \frac{g^* + \sum_{k=1}^{K}\lambda_k \min\{0, c(0) - R_k\}}{\mu} = 0
\end{aligned}$$

and then $c(1) < c(0) \le 0$, a contradiction to $c(1) \ge 0$. Therefore, $c(n) \le c(n+1)$ for all n. □

Property 3.2: The optimal CAC policy always accepts calls from the class with the highest lump-sum revenue whenever there are available resources.

Proof: Without loss of generality, let class 1 be the class that has the highest lump-sum revenue, i.e., $R_1 \ge R_k$ for all k.

Assume towards a contradiction that in some state $n < C$, the optimal CAC policy blocks class-1 calls, i.e., $c(n) > R_1$. From *Lemma 3.1*, $c(C-1) \ge c(n) > R_1 \ge R_k$ for all k. Since $c(C) = \infty$, from (26), we have

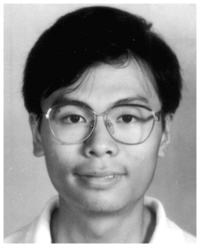
$$\begin{aligned}
c(C-1) &= \frac{g^* + \sum_{k=1}^{K}\lambda_k \min\{0, c(C) - R_k\}}{C\mu} = \frac{g^*}{C\mu}\\
c(C-2) &= \frac{g^* + \sum_{k=1}^{K}\lambda_k \min\{0, c(C-1) - R_k\}}{(C-1)\mu}\\
&= \frac{g^*}{(C-1)\mu} > \frac{g^*}{C\mu} = c(C-1)
\end{aligned}$$

a contradiction to *Lemma 3.1*. Therefore, in all state $n < C$, i.e., when the system has available resources, the optimal CAC policy should accept calls from the class with the highest lump-sum revenue. □

REFERENCES

[1] E. Aarts and J. K. Lenstra, Eds., *Local Search in Combinatorial Optimization*. New York: Wiley, 1997.
[2] J. M. Akinpelu, "The overload performance of engineering networks with nonhierarchical and hierarchical routing," *AT&T Tech. J.*, vol. 63, pp. 1261–1281, 1984.
[3] E. Altman, T. Jimenez, and G. Koole, "On optimal call admission control in a resource-sharing system," *IEEE Trans. Commun.*, vol. 49, no. 9, pp. 1659–1668, Sep. 2001.
[4] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 2nd ed. Belmont, MA: Athena Scientific, 2001, vol. 2.
[5] D. Y. Burman, J. P. Lehoczky, and Y. Lim, "Insensitivity of blocking probabilities in a circuit-switching network," *J. Appl. Prob.*, vol. 21, pp. 850–859, 1984.
[6] Z. Dziong and L. G. Mason, "Fair-efficient call admission control policies for broadband networks—A game theoretic framework," *IEEE/ACM Trans. Netw.*, vol. 4, no. 1, pp. 123–136, Feb. 1996.

[7] G. J. Foschini, B. Gopinath, and J. F. Hayes, "Optimum allocation of servers to two types of competing customers," *IEEE Trans. Commun.*, vol. COM-29, no. 7, pp. 1051–1055, Jul. 1981.
[8] A. Gavious and Z. Rosberg, "A restricted complete sharing policy for a stochastic knapsack problem in B-ISDN," *IEEE Trans. Commun.*, vol. 42, no. 7, pp. 2375–2379, Jul. 1994.
[9] D. Hong and S. S. Rappaport, "Traffic model and performance analysis for cellular mobile radio telephone systems with prioritized and nonprioritized handoff procedures," *IEEE Trans. Veh. Technol.*, vol. VT-35, no. 1, pp. 77–99, Jan. 1986.
[10] J. S. Kaufman, "Blocking in a shared resource enviroment," *IEEE Trans. Commun.*, vol. COM-29, no. 10, pp. 1474–1481, Oct. 1981.
[11] F. P. Kelly, "Routing and capacity allocation in networks with trunk reservation," *Math. Oper. Res.*, vol. 15, no. 4, pp. 771–792, 1990.
[12] ——, "Loss networks," *Ann. Appl. Prob.*, vol. 1, no. 3, pp. 319–378, 1991.
[13] B. Li, C. Lin, and S. T. Chanson, "Analysis of a hybrid cutoff priority scheme for multiple classes of traffic in multimedia wireless netowrks," *ACM/Baltzer J. Wireless Netw.*, vol. 4, pp. 279–290, Aug. 1998.
[14] S. A. Lippman, "Applying a new device in the optimization of exponential queueing systems," *Oper. Res.*, vol. 23, pp. 687–710, 1975.
[15] B. L. Miller, "A queueing reward system with several customer classes," *Manage. Sci.*, vol. 16, pp. 234–245, 1969.
[16] J. Ni, D. H. K. Tsang, S. Tatikonda, and B. Bensaou, "Threshold and reservation based call admission control policies for multiservice resource-sharing systems," in *Proc. IEEE INFOCOM*, Mar. 2005, vol. 2, pp. 773–783.
[17] T. Oda and Y. Watanabe, "Optimal trunk reservation for a group with multislot traffic streams," *IEEE Trans. Commun.*, vol. 38, no. 7, pp. 1078–1084, Jul. 1990.
[18] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York: Wiley, 1994.
[19] R. Ramjee, P. Nagarajan, and D. Towsley, "On optimal call admission control in cellular networks," in *Proc. IEEE INFOCOM*, Mar. 1996, vol. 1, pp. 43–50.
[20] K. W. Ross and D. H. K. Tsang, "The stochastic knapsack problem," *IEEE Trans. Commun.*, vol. 37, no. 7, pp. 740–747, Jul. 1989.
[21] K. W. Ross and D. H. K. Tsang, "Optimal circuit access policies in an ISDN environment: A Markov decision approach," *IEEE Trans. Commun.*, vol. 37, no. 9, pp. 934–939, Sep. 1989.
[22] K. W. Ross, *Multiservice Loss Models for Broadband Telecommunication Networks*. New York: Springer, 1995.
[23] S. M. Ross, *Stochastic Processes*, 2nd ed. New York: Wiley, 1996.
[24] D. H. K. Tsang and K. W. Ross, "Algorithms to determine exact blocking probabilities for multirate tree networks," *IEEE Trans. Commun.*, vol. 38, no. 8, pp. 1266–1271, Aug. 1990.
[25] O. T. W. Yu and V. C. M. Leung, "Adaptive resource allocation for prioritized call admission over an ATM-based wireless PCN," *IEEE J. Sel. Areas Commun.*, vol. 15, no. 7, pp. 1208–1225, Sep. 1997.



Jian Ni (S'02) received the B.Eng. degree in automation from Tsinghua University, Beijing, China, in 2001, and the M.Phil. degree in electrical and electronic engineering from the Hong Kong University of Science and Technology (HKUST), Hong Kong, in 2003. He is currently working toward the Ph.D. degree in electrical engineering at Yale University, New Haven, CT.

His research interests include performance analysis, protocol/algorithm design, inference, control, and optimization of computer and communication networks.

Danny H. K. Tsang (M'82–SM'00) was born in Hong Kong. He received the B.Sc. degree in mathematics and physics from the University of Winnipeg, Winnipeg, MB, Canada, in 1979, the B.Eng. and M.A.Sc. degrees, both in electrical engineering, from the Technical University of Nova Scotia, Halifax, NS, Canada, in 1982 and 1984, respectively, and the Ph.D. degree in electrical engineering from the University of Pennsylvania, Philadelphia, in 1989.

He joined the Department of Mathematics, Statistics and Computing Science, Dalhousie University, Halifax, NS, Canada, in 1989, where he was an Assistant Professor in the computing science division. He joined the Department of Electrical and Electronic Engineering, Hong Kong University of Science and Technology, Hong Kong in 1992, where he is currently an Associate Professor. His current research interests include congestion controls in broadband networks, wireless LANs, Internet QoS and applications, application-level multicast and overlay networks, and optical networks. During his leave from HKUST in 2000–2001, Dr. Tsang assumed the role of Principal Architect at Sycamore Networks in the United States. He was responsible for the network architecture design of Ethernet MAN/WAN over SONET/DWDM networks. He invented the 64 B/65 B encoding and also contributed to the 64 B/65 B encoding proposal for Transparent GFP in the ITU T1X1.5 standard.

Dr. Tsang was the General Chair of the IFIP Broadband Communications'99 held in Hong Kong. He also received the Outstanding Paper from Academe Award at the IEEE ATM Workshop'99. He is currently an Associate Editor for *OSA Journal of Optical Networking*, an Associate Technical Editor for the *IEEE Communications Magazine*, and a TPC member for IEEE BroadNets'05, HSPR'05, GI'05, ICC'05, VTC'06, and INFOCOM'06.

Sekhar Tatikonda (S'92–M'00) received the Ph.D. degree in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, in 2000.

From 2000 to 2002, he was a Postdoctoral Fellow in the Computer Science Department, University of California, Berkeley. He is currently an Assistant Professor of Electrical Engineering at Yale University, New Haven, CT. His research interests include communication theory, information theory, stochastic control, distributed estimation and control, statistical machine learning, and inference.

Brahim Bensaou (M'97) received the B.Sc. degree in computer science from the University of Science and Technology of Algiers, Algeria, in 1987, the D.E.A. degree from the University Paris XI, Orsay, France, in 1988, and the Doctoral degree in computer science in 1993 from the University Paris VI, Paris, France.

He spent four years as a Research Assistant with France Telecom's Center National d'Etudes des Télécommunications, known now as France Telcom R&D, while working toward the Ph.D. degree, on queueing models and performance evaluation of ATM networks. He then spent two years as a Research Associate with the Department of Electrical and Electronic Engineering, Hong Kong University of Science and Technology (HKUST), Hong Kong, from 1995 to 1997, working on various problems in traffic and congestion control in high-speed broadband networks. In 1997, he joined the Center for Wireless Communications, Singapore (known now as the Institute for Infocomm Research), where he was a Senior Member of Technical Staff leading the networking research group. Since fall 2000, he has been on the faculty of HKUST as an Assistant Professor in Computer Science. His research interests are primarily in quality of service in computer communication networks, with current focus on wireless and mobile networking, in particular, medium access control, scheduling, traffic control, congestion control, and QoS routing in wireless and mobile ad-hoc networks.

Dr. Bensaou is a member of the IEEE Communications Society, the ACM, and the ACM SIGCOMM, has been on many IEEE-sponsored conference program committees, and has reviewed extensively for various IEEE transactions.