An Inversion Algorithm to Compute Blocking Probabilities in Loss Networks With State-Dependent Rates

Gagan L. Choudhury,[1] Kin K. Leung,[2] and Ward Whitt[3]

IEEE/ACM Transactions on Networking 3 (1995) 585–601

***Abstract*— We extend our recently developed algorithm for computing (exact) steady-state blocking probabilities for each class in product-form loss networks to cover general state-dependent arrival and service rates. This generalization allows us to consider, for the first time, a wide variety of buffered and unbuffered resource-sharing models with non-Poisson traffic, as may arise with overflows in the context of alternative routing. As before, we consider non-complete-sharing policies involving upper-limit and guaranteed-minimum bounds for the different classes, but here we consider both bounds simultaneously. These bounds are important for providing different grades of service with protection against overloads by other classes. Our algorithm is based on numerically inverting the generating function of the normalization constant, which we derive here. Major features of the algorithm are: dimension reduction by elimination of non-binding resources and by conditional decomposition based on special structure, an effective scaling algorithm to control errors in the inversion, efficient treatment of multiple classes with identical parameters and truncation of large sums. We show that the computational complexity of our inversion approach is usually significantly lower than the alternative recursive approach.**

1 Introduction

In [6, 8] we developed a new algorithm for solving product-form models based on numerically inverting the generating function of the normalization constant. Here we extend the algorithm to cover loss networks (or resource-sharing models) with general *state-dependent arrival and service rates.* The model has multiple *resources*, each containing multiple resource *units* which provide service to multiple *job classes.* Each job requires a number of units from each resource, which may be zero, one or greater than one. In a circuit-switched telecommunications network, the resources may be links, the resource units may be circuits on these links, and the jobs may be calls.

In the standard loss network model [3, 6, 10, 11, 12, 15, 21, 23, 28, 31, 32], the job arrival processes are independent Poisson processes and the job holding times are assumed to be independent with exponential distributions having a mean depending on the job class. (The exponential assumption can be relaxed by virtue of insensitivity [3].) Here, however, we consider a more general model with state-dependent arrival and service rates, which still has a product-form steady-state distribution. In particular, we assume that the vector representing the numbers of jobs in service of each class evolves as a continuous-time Markov chain, with the arrival and service rates of each class depending on the number of jobs from that class already in service. The state-dependent arrival rates may be used in two ways: First, the arrival rate may truly be state-dependent, as when there are only finitely many sources or when the arrival rate is controlled based on the number of jobs in service. Second, the state-dependent arrival rate may be introduced to approximate non-state-dependent non-Poisson traffic (explained in Section 5), generalizing Delbrouck's [13, 14] treatment of a more elementary model.

The state-dependent service rate includes as a special case the standard unbuffered model in which each job goes into service immediately upon entering the system, and hence the service rate for a class is proportional to the number of jobs of that class in service. However, the main attraction of the state-dependent service rate is that it allows us to model the rich class of buffered resource-sharing models in which a job is buffered upon entering the system and starts service only after other jobs of its class have finished their service. The number of servers per class may be one or more. In contrast to the unbuffered variant, where the resource units are servers, in the buffered variant the resource units are buffer spaces. In the special case of unlimited servers per class the buffered variant is identical to the unbuffered variant.

The standard loss model has a *complete-sharing* (CS) policy, in which jobs are admitted whenever all the required resource units are free. Here, however, we consider more general resource-sharing policies involving extra linear constraints (which make the state space coordinate convex [17]). We pay particular attention to the case in which *upper-limit* (UL) and *guaranteed-minimum* (GM) bounds are assigned to each class. The UL bounds limit the number of jobs from each class that can be in service. The GM bound guarantees that there is always space for a specified number of jobs from each class. A set of GM bounds is equivalent to an upper limit on the resource units used by each subset of the classes. The UL and GM bounds are equivalent for two classes, but not for more than two classes. We focus on *combined UL and GM bounds* (which cannot be reduced to either one alone). The UL and GM bounds are very useful for providing protection against overloads and for providing different grades of service to different job classes.

A version of this paper was presented at *IEEE Infocom '95*, Boston, MA, April 1995 [7].
[1]AT&T Bell Laboratories, Room 1L-238, Holmdel, NJ 07733-3030 (gagan@buckaroo.att.com).
[2]AT&T Bell Laboratories, Room 1L-215, Holmdel, NJ 07733-3030 (kin-buckaroo.att.com).
[3]AT&T Bell Laboratories, Room 2C-178, Murray Hill, NJ 07974-0636 (wow@research.att.com).

If all requirements under the sharing policy in use can be met upon arrival of a new job, then the new job is admitted, and all required resource units are held throughout the job holding time. Otherwise, the job is blocked and lost. The primary measures of performance are the blocking probabilities of the different classes.

The basic model we consider assumes fixed routing. However, we can also treat alternative routing approximately, extending [14, 15], by using state-dependent arrival rates represent overflow traffic associated with alternative routing. Networks of moderate size or with special structure allowing dimension reduction (see [6, 8]) can be treated by our method exactly. Large networks without special structure can be analyzed approximately by extending the reduced-load fixed-point approximations [21].

The fact that the generalizations we introduce have product-form is easy to show. So our main contribution is to provide an effective algorithm for computing the normalization constants in these models. We also show that blocking probabilities and other steady-state characteristics have simple expressions in terms of normalization constants. In general, a steady-state performance measure (e.g., a moment) may involve computation of a very large number of normalization constants, but we show in Section 5 that it is always possible to express the quantity of interest in terms of a small number of modified normalization constants, which are as easy to compute by our method as the standard normalization constant.

As in [6, 8], our algorithm is based on deriving a convenient expression for the generating function of the normalization constant and then numerically inverting the generating function. We use the Fourier-series method for inverting generating functions [1, 2, 9]. Since the basic algorithm is already described in [1, 2, 6, 8, 7, 9], we will be brief here, concentrating on new features. It is well known that computing the normalization constant is a challenging problem when the model is large. Through numerical examples we show that the numerical inversion approach is remarkably effective.

The numerical inversion algorithm has a number of computational advantages: First, large finite sums may be efficiently computed through judicious *truncation* or through *acceleration methods.* Second, for large models with a high-dimensional generating function, it is often possible to *reduce the effective dimension* by first, *eliminating nonbinding resources* and, second, by performing *conditional decomposition*, i.e., by inverting the variables in a good order. For example, dimension reduction enables us to solve models with UL and GM resource-sharing policies nearly as quickly as the standard model with the CS sharing policy. Similar approaches to dimension reduction (but with quite different algorithms) have been used by Lam and Lien [24] for closed queueing networks and by Conway, Pinsky and Tripandapani [11, 12, 28] for special cases of the loss networks considered here. It is also possible to reduce the computations by exploiting *multiplicities*, i.e., multiple classes with identical parameters. We can make our models much larger by increasing multiplicities at negligible computational cost. In models for large systems the multiplicities occur very naturally.

In addition to developing the inversion algorithm, we also derive a new convolution-based recursive algorithm for the most general model considered in this paper. However, we also show that the recursive algorithms are much slower than our inversion-based algorithms, and so their primary use here is in verifying the inversion-based computations for small models. In the past, recursive algorithms have been derived from generating functions (for different models) by Reiser and Kobayashi [29], Delbrouck [14], Kogan and Shenfild [22], Mitra and Morrison [26] and Morrison [27].

We now discuss the unbuffered and buffered variants separately in more detail.

1.1 The Unbuffered Variant

The unbuffered variant has become widely recognized as a fundamental model for communication networks. For instance, it is now being considered to analyze the performance of wireless networks [34] and emerging high-speed communication networks employing the asynchronous transfer mode (ATM) technology [26]. For ATM systems, the unbuffered variant of loss models has possible applications at both the call level and the burst level. In the basic application, the resources are the bandwidth available at the network facilities. This model applies at the call level if we can assign an effective bandwidth requirement to each call (on each link). The loss network applies at the burst level if we can assign an effective bandwidth requirement to each burst within an established connection. The possibility of assigning such effective bandwidths and ways to do so are actively being studied, e.g., see [36].

In the standard loss network model each arrival process is a Poisson process, but it is desirable to generalize the model in order to represent arrival processes that are significantly more or less bursty than the Poisson process. For this purpose, Delbrouck [13, 14] and Dziong and Roberts [15] considered linear state-dependent arrivals, the so-called *Bernoulli-Poisson-Pascal* (BPP) *model.* The less bursty binomial case is also directly of interest because it corresponds to arrivals from finitely many sources. For practical applications, it is important that the two parameters α_j and β_j in the state-dependent arrival-rate function for class j, $\lambda_j(k) \equiv \alpha_j + \beta_j k$, where k class-j jobs are in service, can be conveniently expressed in terms of the overall arrival rate and peakedness. (The peakedness is a familiar partial characterization of burstiness.) Hence, the BPP model is relatively easy to apply to represent non-Poisson arrival processes, as arise in overflow processes occurring with alternative routing; see Section 5.

One of our contributions here is to develop faster algorithms (see Section 8). However, a more important contribution is to be able to solve the model when there are the non-CS resource-sharing policies involving UL and GM parameters. Previous algorithms for non-CS resource sharing have been very limited. A recursive algorithm for the UL policy with one or two resources was developed by Chuah [10]. For a single resource, the UL policy is equivalent to the tree networks considered by Tsang and Ross [31] (for the case of Poisson arrivals).

1.2 The Buffered Variant

The buffered variant with a single constant-rate server per class and a single resource was considered by Kamoun and Kleinrock [16] to analyze a node of a store-and-forward computer network, where the outgoing channels of the node share a certain number of buffers. Each job class corresponds to the traffic destined to a particular outgoing channel. The resource here is the buffer space. As before, the job holding time is the period the job occupies the resource. In this case it is the waiting time plus the service time.

Unlike the unbuffered variant where almost all prior work considers only the CS policy, Kamoun and Kleinrock did consider the UL, GM and combined UL and GM policies. However, we generalize their work in several ways: First, we allow multiple servers per class. Second, they assumed a single resource, while we consider multiple resources. The multiple resources could either be multiple nodes of a computer network or more than one resource at a single node (e.g., several types of storage elements). Kamoun and Kleinrock assumed that each job holds a single buffer element but we can allow each job to hold multiple and possibly different numbers of buffer elements. Kamoun and Kleinrock required the different job classes to have either all identical traffic intensities or all different traffic intensities. We do not have this restriction. For the more complex UL, GM and combined UL and GM policies, Kamoun and Kleinrock had further restrictions on the system parameters. Also in some cases it appears that the computational complexity of their algorithm grows exponentially with r, the number of job classes. (Their numerical examples are only for 2 classes.) By contrast, we do not have any restrictions on the system parameters and our computational complexity grows linearly with the number of different job classes and does not grow at all if the parameters of a new job class are identical to those of one of the existing classes, thereby allowing us to consider a very large number of job classes. With all these generalizations, we believe that we have made an important contribution to analyzing buffered resource-sharing models, which are important for modeling high-speed network buffers.

1.3 Organization of the Paper

Here is how the rest of this paper is organized. In Section 2 we specify the model and derive the generating function of the normalization constants with the complete-sharing policy. In Section 3 we consider non-complete-sharing policies. In Section 4 we discuss how to compute blocking probabilities. In Section 5 we show how to compute probability distributions and moments for each class. In Section 6 we discuss modeling with BPP arrival processes (the connection to peakedness). In Section 7 we introduce a new method for reducing the dimension of the generating function by eliminating very lightly loaded resources from the model before doing the inversion. In Section 8 we describe a new scaling algorithm that allows accurate inversion of all generating functions considered in this paper. In Section 9 we discuss multiplicities. In Section 10 we discuss the computational complexity of our algorithm. In Section 11 we show how the generating functions can be used to derive recursive algorithms. Finally, in Sections 12–14 we present illustrative numerical examples.

To save space, some material in the conference version of this paper [7] has been omitted here. In Section 8 of [7] we show how to calculate derivatives of the blocking probabilities with respect to model parameters by inversion. The single-source examples in Sections 9 and 10 of [7] are different from the examples here. The example in Section 9 of [7] is the classical resource-sharing model (single resource) with the CS policy and a finite-source input. We implemented the recursive algorithm of Delbrouck [14] and the uniform asymptotic approximation (UAA) of Mitra and Morrison [26] and used them to validate the inversion algorithm. (A variant of UAA was also developed by Kogan and Shenfild [22].) The example in Section 10 of [7] contains all the model variations considered in this paper in the context of a single resource. In particular, it has four types of classes, both buffered and unbuffered, with state-dependent arrival rates and non-CS policies. Up to 4000 classes share up to 100,000 resource units. Exploiting multiplicities and truncation, we are able to solve the largest case in only a few seconds on a SUN SPARC-2 workstation.

After completing this paper, we extended the algorithm here in [5] to cover the case of state-dependent arrival of batches. Previous work on batches was done by van Doorn and Panken [33], Kaufman and Rege [18] and Morrison [27].

2 Complete Sharing

2.1 The General Case

Consider a loss network with p *resources* and r *classes of jobs.* Let the resources be indexed by i and the job classes by j. Let resource i have K_i *units*, $1 \leq i \leq p$, and let $\mathbf{K} \equiv (K_1, \ldots, K_p)$ be the *capacity vector.* (We let vectors be either row vectors or column vectors; it will be clear from the context.) Each class j job requires a_{ij} units on resource i, where a_{ij} is a (deterministic) nonnegative integer. Let $\mathbf{A}$ be the $p \times r$ *requirements matrix* with elements a_{ij}.

Let the system *state vector* be $\mathbf{n} \equiv (n_1, \ldots, n_r)$, where n_j is the number of class j jobs currently in process. Let $S_P(\mathbf{K})$ be the set of allowable states, which depends on the capacity vector $\mathbf{K}$ and the *sharing policy* P. The state space $S_P(\mathbf{K})$ is a subset of $\mathbf{Z}_+^r$, the r-fold product of the nonnegative integers. With non-complete-sharing policies, the set of allowable states will typically depend on other parameters besides $\mathbf{K}$. For the complete-sharing policy,

$$S_{CS}(\mathbf{K}) = \{\mathbf{n} \in \mathbf{Z}_+^r : \mathbf{A}\mathbf{n} \le \mathbf{K}\} \ . \tag{2.1}$$

The stochastic process $\{\mathbf{N}(t) : t \ge 0\}$, where $\mathbf{N}(t)$ gives the system state at time t, is an irreducible finite-state continuous-time Markov chain (CTMC) with a unique steady-state probability vector π. If there are k class-j jobs in the network, then the arrival rate of class-j jobs is $\lambda_j(k)$. Let $\mu_j(k)$ be the rate of class-j service completion when there are k class-j jobs in the system. In the unbuffered variant $\mu_j(k) = k\mu_j$ and in the buffered variant with s_j servers for class j, $\mu_j(k) = k\mu_j$ for $k \le s_j$ and $\mu_j(k) = s_j\mu_j$ for $k > s_j$. Each job is admitted if all desired resource units can be provided; otherwise the job is blocked and lost (without affecting future arrivals). All resource units used by a job are released at the end of the job holding time.

The steady-state probability vector has the simple *product form*

$$\pi(\mathbf{n}) = g(\mathbf{K})^{-1} f(\mathbf{n}), \tag{2.2}$$

where

$$f(\mathbf{n}) = \prod_{j=1}^{r} f_j(n_j), \ \ f_j(n_j) = \Lambda_j(n_j)/M_j(n_j) \tag{2.3}$$

$$\Lambda_j(n_j) = \prod_{k=0}^{n_j-1} \lambda_j(k), \quad M_j(n_j) = \prod_{k=1}^{n_j} \mu_j(k), \tag{2.4}$$

and the *normalization constant* (or *partition function*) is

$$g(\mathbf{K}) \equiv g_P(\mathbf{K}) = \sum_{\mathbf{n} \in S_P(\mathbf{K})} f(\mathbf{n}). \tag{2.5}$$

In the unrestricted case (without capacity constraints), $\mathbf{N}(t)$ is a vector of independent birth-and-death processes and thus a reversible Markov process. Thus, the restricted process is also a reversible Markov process with a steady-state distribution that is simply a truncation and renormalization of the distribution in the unrestricted case; see Section 1.6 of [19].

We now obtain the generating function of $g(\mathbf{K})$ in the case of a CS-policy. By definition

$$G(\mathbf{z}) = \sum_{K_1=0}^{\infty} \cdots \sum_{K_p=0}^{\infty} g(\mathbf{K}) z_1^{K_1} \cdots z_p^{K_p} \tag{2.6}$$

for a vector of complex variables $\mathbf{z} = (z_1, \ldots, z_p)$. We obtain a more compact expression by changing the order of summation. For this purpose, let $\overline{K}_i = \sum_{j=1}^r a_{ij} n_j$. Then

$$\begin{aligned} G(\mathbf{z}) &= \sum_{n_1=0}^{\infty} \cdots \sum_{n_r=0}^{\infty} \ \sum_{K_1=\overline{K}_1}^{\infty} \cdots \sum_{K_p=\overline{K}_p}^{\infty} f(\mathbf{n}) z_1^{K_1} \cdots z_p^{K_p} \\ &= \prod_{i=1}^{p} (1-z_i)^{-1} \sum_{n_1=0}^{\infty} \cdots \sum_{n_r=0}^{\infty} \ \prod_{j=1}^{r} \left(f_j(n_j) \prod_{i=1}^{p} z_i^{a_{ij} n_j} \right) \\ &= \prod_{i=1}^{p} (1-z_i)^{-1} \prod_{j=1}^{r} G_j(\mathbf{z}), \end{aligned} \tag{2.7}$$

where

$$G_j(\mathbf{z}) = \sum_{n_j=0}^{\infty} f_j(n_j) \prod_{i=1}^{p} z_i^{a_{ij} n_j}. \tag{2.8}$$

From (2.7), we see that the transform factors into r terms, one for each class. However, in general, the factors $G_j(\mathbf{z})$ in (2.8) will have common z_i variables. In this section we do not make any further assumption on the arrival and service rates; hence no simplification of (2.8) is possible. However, the infinite series in (2.8) may always be truncated by realizing that $n_j \le \min_i \lfloor K_i/a_{ij} \rfloor \equiv N_j$. So we can set $\lambda_j(n_j) = 0$ for $n_j \ge N_j$ which also implies $f_j(n_j) = 0$ for $n_j \ge N_j$. This gives

$$G_j(\mathbf{z}) = \sum_{n_j=0}^{N_j} f_j(n_j) \prod_{i=1}^{p} z_i^{a_{ij} n_j}. \tag{2.9}$$

Using (2.9) we can do computations for arbitrary state-dependent arrival and service rates. This is more general than the models to be considered in Sections 2.2 and 2.3 where (2.8) has a closed-form expression.

2.2 The Unbuffered Variant With BPP Arrivals

In this case $M_j(n_j) = \mu_j^{n_j} n_j!$. For Poisson arrivals, $\lambda_j(k) = \lambda_j$. Here

$$f_j(n_j) = \left(\frac{\lambda_j}{\mu_j}\right)^{n_j} \cdot \frac{1}{n_j!} = \frac{\rho_j^{n_j}}{n_j!} \tag{2.10}$$

where $\rho_j = \lambda_j/\mu_j$. Combining (2.8) and (2.10) yields

$$G_j(\mathbf{z}) = \exp\left[\rho_j \prod_{i=1}^{p} z_i^{a_{ij}}\right], \tag{2.11}$$

which is the same as (2.12) of [6].

If, instead, $\lambda_j(k) = \alpha_j + \beta_j k$ where $\beta_j \ne 0$, as in the binomial and Pascal (negative binomial) cases of the BPP model of [13, 14, 15], then

$$f_j(n_j) = \binom{r_j + n_j - 1}{r_j - 1} \left(\frac{\beta_j}{\mu_j}\right)^{n_j} \tag{2.12}$$

where $r_j = \alpha_j/\beta_j$. Combining (2.8) and (2.12) yields

$$\begin{aligned} G_j(\mathbf{z}) &= \sum_{n_j=0}^{\infty} \binom{r_j + n_j - 1}{r_j - 1} \left(\frac{\beta_j}{\mu_j} \prod_{i=1}^{p} z_i^{a_{ij}}\right)^{n_j} \\ &= \left(1 - \frac{\beta_j}{\mu_j} \prod_{i=1}^{p} z_i^{a_{ij}}\right)^{-r_j}. \end{aligned} \tag{2.13}$$

With infinite state spaces, we would need to assume that $\beta_j < \mu_j$ in order to have a proper steady-state distribution, but we can allow $\beta_j \ge \mu_j$ because we have a finite state space.

In (2.13) we can allow β_j to be negative provided that $\lambda_j(k) \equiv \alpha_j + \beta_j k = 0$ for some k. The case of β_j negative includes the finite-source input case. When there are N_j

sources for class j, each with arrival rate λ'_j, $\alpha_j = N_j\lambda'_j$, $\beta_j = -\lambda'_j$ and $r_j \equiv \alpha_j/\beta_j = -N_j$. Further, defining $p_j = \lambda'_j/(\lambda'_j + \mu_j)$, (2.13) becomes

$$G_j(\mathbf{z}) = \left(1 - p_j + p_j \prod_{i=1}^{p} z_i^{a_{ij}}\right)^{N_j} (1-p_j)^{N_j}. \qquad (2.14)$$

2.3 The Buffered Variant With Poisson Arrivals

Let s_j represent the number of servers for class j, μ_j the service rate per server, and λ_j the arrival rate. Let $\rho_j = \lambda_j/\mu_j$. Then

$$f_j(n_j) = \begin{cases} \rho_j^{n_j}/n_j! & \text{for} \quad n_j < s_j \\ \rho_j^{s_j}/s_j!(\rho_j/s_j)^{n_j - s_j} & \text{for} \quad n_j \geq s_j. \end{cases} \qquad (2.15)$$

Combining (2.8) with (2.15), we get

$$G_j(\mathbf{z}) = \sum_{n_j=0}^{s_j-1} \frac{\left(\rho_j \prod_{i=1}^{p} z_i^{a_{ij}}\right)^{n_j}}{n_j} + \frac{\left(\rho_j \prod_{i=1}^{p} z_i^{a_{ij}}\right)^{s_j}}{s_j} \left(1 - \rho_j \prod_{i=1}^{p} z_i^{a_{ij}}/s_j\right)^{-1}. \qquad (2.16)$$

As s_j approaches infinitely, (2.16) approaches (2.11). For $s_j = 1$,

$$G_j(\mathbf{z}) = \left(1 - \rho_j \prod_{i=1}^{p} z_i^{a_{ij}}\right)^{-1}. \qquad (2.17)$$

It is interesting to note that the buffered variant with single server per class and Poisson arrivals is the same as the unbuffered variant with BPP arrivals, $r_j = 1$ and $\beta_j/\mu_j = \rho_j$.

3 Other Sharing Policies

We can introduce other sharing policies by imposing additional constraints on the set of feasible states. As noted in [6], each additional linear constraint is equivalent to adding another resource. Resource i results in the constraint $\sum_{j=1}^{r} a_{ij}n_j \leq K_i$ where K_i and a_{ij} are nonnegative integers. Assuming that a new constraint is expressed in terms of rational numbers, it can be re-expressed in terms of integers.

Hence, we can add linear constraints without changing the general form of the model, but the computational complexity is exponential in the number of resources. Therefore, it is significant that certain special extra sets of linear constraints can be treated efficiently. As shown in [6], this is true for the *upper limit* (UL) and *guaranteed minimum* (GM) sharing policies. (For GM, we require special structure.) In both cases an extra linear constraint is added for each class, but the effective dimension of the generating function after dimension reduction increases by at most 1.

In this paper we show that we can consider the *combined* UL and GM policy with only slightly more computation than the CS policy. (Clearly the individual UL and GM policies are special cases.) As in [6], we impose an additional condition to treat the GM policy. (This condition can be removed for the pure UL policy.) In particular, we assume that a_{ij} is either b_j or 0 for all i. We let $\delta_{ij} = 1$ if $a_{ij} > 0$ and $\delta_{ij} = 0$ otherwise. Let N_j be the number of resource *units* guaranteed for class j jobs (which must be the same for each resource type) and let $\mathbf{N} = (N_1, \ldots, N_r)$.

Let L_{ij} be the upper limit on the number of resource units of type i that class j jobs are allowed to use simultaneously. Let M_j be the minimum value of $\lfloor L_{ij}/a_{ij} \rfloor$ over all i, where $\lfloor x \rfloor$ is the greatest integer less than or equal to x. Let $\mathbf{M} = (M_1, \ldots, M_r)$.

The state space for sharing with both UL and GM bounds, which we denote by UG, is the intersection of the two separate state spaces, i.e.,

$$S_{UG}(\mathbf{K}, \mathbf{M}, \mathbf{N}) = S_{UL}(\mathbf{K}, \mathbf{M}) \cap S_{GM}(\mathbf{K}, \mathbf{N}), \qquad (3.1)$$

where

$$S_{UL}(\mathbf{K}, \mathbf{M}) = \{\mathbf{n} \in \mathbf{Z}_+^r : \mathbf{An} \leq \mathbf{K}, \ \mathbf{n} \leq \mathbf{M}\} \qquad (3.2)$$

and

$$S_{GM}(\mathbf{K}, \mathbf{N}) = \left\{\mathbf{n} \in \mathbf{Z}_+^r : \sum_{j=1}^{r} (a_{ij}n_j \vee \delta_{ij}N_j) \leq K_i, \ 1 \leq i \leq p\right\} \qquad (3.3)$$

with $x \vee y = \max\{x, y\}$. From (3.3), we see that GM bounds for r classes corresponds to 2^r linear constraints, one of which is the CS constraint and another of which is the GM consistency condition $\sum_{j=1}^{r} N_j \leq K_i$. In other words, there is a linear constraint corresponding to each nonempty subset of classes.

In the general case, the generating function of the normalization constant $g(\mathbf{K}, \mathbf{M}, \mathbf{N})$ is

$$G(\mathbf{z}, \mathbf{y}, \mathbf{x}) = \sum_{K_1=0}^{\infty} \cdots \sum_{N_r=0}^{\infty} g(\mathbf{K}, \mathbf{M}, \mathbf{N}) \left[z_1^{K_1} \cdots z_p^{K_p} \times y_1^{M_1} \cdots y_r^{M_r} x_1^{N_1} \cdots x_r^{N_r}\right]. \qquad (3.4)$$

After changing the order of summation, we obtain

$$G(\mathbf{z}, \mathbf{y}, \mathbf{x}) = \sum_{n_1=0}^{\infty} \cdots \sum_{n_r=0}^{\infty} \sum_{N_1=0}^{\infty} \cdots \sum_{N_r=0}^{\infty} \sum_{M_1=n_1}^{\infty} \cdots \sum_{M_r=n_r}^{\infty} \sum_{K_1=K_1(\mathbf{n},\mathbf{N})}^{\infty} \cdots \sum_{K_p=K_p(\mathbf{n},\mathbf{N})}^{\infty} \left[\prod_{j=1}^{r} f_j(n_j) x_j^{N_j} y_y^{M_j} \left(\prod_{i=1}^{p} z_i^{K_i}\right)\right] \qquad (3.5)$$

for $K_i(\mathbf{n}, \mathbf{N}) = \sum_{j=1}^{r} (a_{ij}n_j \vee \delta_{ij}N_j)$. Hence,

$$G(\mathbf{z}, \mathbf{y}, \mathbf{x}) = \prod_{i=1}^{p} (1 - z_i)^{-1} \prod_{j=1}^{r} G_j(\mathbf{z}, \mathbf{y}, \mathbf{x}), \qquad (3.6)$$

where

$$G_j(\mathbf{z},\mathbf{y},\mathbf{x}) = (1-y_j)^{-1}\left[\left(1-x_j\prod_{i=1}^{p} z_i^{\delta_{ij}}\right)^{-1} F_j\left(y_j x_j^{b_j}\prod_{i=1}^{p} z_i^{\delta_{ij}b_j}\right) + (1-x_j)^{-1}F_j\left(y_j\prod_{i=1}^{p} z_i^{a_{ij}}\right) - (1-x_j)^{-1}F_j\left(y_j x_j^{b_j}\prod_{i=1}^{p} z_i^{a_{ij}}\right)\right] \quad (3.7)$$

and

$$F_j(x) \equiv \sum_{j=0}^{\infty} f_j(n_j)x^{n_j}. \quad (3.8)$$

Since $G_j(\mathbf{z},\mathbf{y},\mathbf{x})$ in (3.7) contains only the variables $\mathbf{z}$, y_j and x_j, the effective dimension in (3.6) can always be reduced from $p+2r$ to $p+2$. However, we can do even better by explicit inversion, as we now show. Explicit inversion with respect to x_j yields

$$G_j(\mathbf{z},y_j,N_j) = (1-y_j)^{-1}\left[\left(\prod_{i=1}^{p} z_i^{\delta_{ij}}\right)^{N_j}\sum_{n_j=0}^{\lfloor N_j/b_j\rfloor} f_j(n_j)y_j^{n_j} + \sum_{n_j=\lfloor N_j/b_j\rfloor+1}^{\infty} f_j(n_j)\left(y_j\prod_{i=1}^{p} z_i^{a_{ij}}\right)^{n_j}\right]. \quad (3.9)$$

Now doing explicit inversion of (3.9) with respect to y_j and remembering that $M_j \geq \lfloor N_j/b_j\rfloor$, yields

$$G_j(\mathbf{z},M_j,N_j) = \left(\prod_{i=1}^{p} z_i^{\delta_{ij}}\right)^{N_j}\sum_{n_j=0}^{\lfloor N_j/b_j\rfloor} f_j(n_j) + \sum_{n_j=\lfloor N_j/b_j\rfloor+1}^{M_j} f_j(n_j)\left(\prod_{i=1}^{p} z_i^{\delta_{ij}}\right)^{n_j b_j}. \quad (3.10)$$

Note the remarkably simple form of (3.10). Assuming N_j to be an integral multiple of b_j, we can rewrite (3.10) as

$$G_j(\mathbf{z},M_j,N_j) = \left(\prod_{i=1}^{p} z_i^{\delta_{ij}}\right)^{N_j}\left[\sum_{l=0}^{\lfloor N_j/b_j\rfloor} f_j(l) + \sum_{l=1}^{M_j-\lfloor N_j/b_j\rfloor} f_j\left(\lfloor N_j/b_j\rfloor + l\right)\left(\prod_{i=1}^{p} z_i^{\delta_{ij}}\right)^{lb_j}\right]. \quad (3.11)$$

The overall remaining generating function is

$$G(\mathbf{z},\mathbf{M},\mathbf{N}) = \prod_{i=1}^{p}(1-z_i)^{-1}\prod_{j=1}^{r} G_j(\mathbf{z},M_j,N_j), \quad (3.12)$$

where $G_j(\mathbf{z},M_j,N_j)$ is given by (3.10) or (3.11). If we use (3.11), then there is a leading term $\prod_{i=1}^{p} z_i^{\sum_{j=1}^{r} N_j\delta_{ij}}$ which can be explicitly taken out, and we can consider a smaller problem with K_i replaced by $K_i - \sum_{j=1}^{r}\delta_{ij}N_j$ for $i = 1,2,\ldots,p$. This step will be especially effective if $K \approx \sum_{j=1}^{r}\delta_{ij}N_j$. As an extreme case, if $K_i = \sum_{j=1}^{r}\delta_{ij}N_j$, $1 \leq i \leq p$, then we get complete partitioning. Then (3.11) and (3.12) provide an explicit expression for the normalization constant as

$$g(\mathbf{K},\mathbf{M},\mathbf{N}) = \prod_{j=1}^{r}\sum_{n_j=0}^{\lfloor N_j/b_j\rfloor} f_j(n_j). \quad (3.13)$$

Using (3.12) and (3.10) or (3.11), we have effectively reduced the effective dimension of inversion from $p+2r$ to p. However, since there are M_j terms in (3.10) and (3.11), the computational complexity is about M_j times that of a closed form p-dimensional inversion. In general, M_j could increase with the K_i, but if there are many classes, M_j may remain small even with large K_i. If, however, M_j is indeed very large, then it will be advantageous to work with $G_j(\mathbf{z},y_j,N_j)$ and do one more level of inversion. If $\lfloor N_j/b_j\rfloor$ is large, then it may even be advantageous to work with $G_j(\mathbf{z},y_j,x_j)$.

All the unbuffered and buffered versions considered in Section 2 are easily obtained by inserting in the corresponding expressions for $f_j(n_j)$ in (3.7), (3.9), (3.10) and (3.11). For the buffered variant (with Poisson arrivals), the sums in (3.10) and (3.11) may be expressed in closed form. Specifically, (3.11) becomes

$$G_j(\mathbf{z},M_j,N_j) = \left(\prod_{i=1}^{p} z_i^{\delta_{ij}}\right)^{N_j}\left[\frac{1-\rho_j^{\lfloor N_j/b_j\rfloor+1}}{1-\rho_j} + \left(1-\rho_j\left(\prod_{i=1}^{p} z_i^{\delta_{ij}}\right)^{b_j}\right)^{-1}\left(\rho_j^{\lfloor N_j/b_j\rfloor+1}\left(\prod_{i=1}^{p} z_i^{\delta_{ij}}\right)^{b_j} - \rho_j^{M_j+1}\left(\prod_{i=1}^{p} z_i^{\delta_{ij}}\right)^{M_j b_j - N_j + b_j}\right)\right]. \quad (3.14)$$

Thus, remarkably, for the buffered variant (with Poisson arrivals), the computation for the combined UL/GM model is just as fast as for the simple CS model. Note that the overall generating function, (2.7) or (3.6), is always a product of factors, with one factor from each class. This property allows us to combine several different types of arrival processes (e.g., from the BPP family), model variants (buffered and unbuffered) and sharing policies (CS and UL/GM) in the same model. We illustrate this capability in our example in Section 10 of [7].

4 Blocking Probabilities

It is important to distinguish between *call (job) blocking* and *time blocking.* Call blocking refers to the blocking experienced by arrivals (which depends on the state at arrival epochs), while time blocking refers to the blocking that would take place at an arbitrary time if there were an arrival at that time (as in the virtual waiting time). Since the steady-state distribution π refers to an arbitrary time, blocking probabilities computed directly from it involve time blocking, but it is not difficult to treat call blocking as well as time blocking. With Poisson arrivals, the two probability distributions at arrival epochs and at an arbitrary time agree, but not more generally; see [25].

The probability that a class-j job would not be admitted at an arbitrary time (time blocking) with a combined UL/GM policy is easily seen to be

$$B_j^{(t)} = 1 - \frac{g(\mathbf{K} - \mathbf{A}\mathbf{e}_j,\ \mathbf{M} - \mathbf{e}_j,\ \mathbf{N} - \mathbf{A}\mathbf{e}_j)}{g(\mathbf{K}, \mathbf{M}, \mathbf{N})}, \qquad (4.1)$$

where $\mathbf{a}_j \equiv (a_{1j}, \ldots, a_{pj})$ is the requirements vector for class j and $\mathbf{e}_j$ is a vector with a 1 in the j^{th} place and 0's elsewhere.

If the blocking probability is very small, then formula (4.1) presents a numerical difficulty, since we are taking the difference of two quantities both much larger than the final answer. Specifically, if we compute in double precision arithmetic with a precision of around 10^{-14}, then large roundoff error will be introduced whenever the blocking probability is near or below 10^{-14}. However, this difficulty may be removed by rewriting (4.1) as

$$B_j^{(t)} = \frac{h(\mathbf{K}, \mathbf{M}, \mathbf{N})}{g(\mathbf{K}, \mathbf{M}, \mathbf{N})}, \qquad (4.2)$$

where

$$\begin{aligned} h(\mathbf{K}, \mathbf{M}, \mathbf{N}) = {} & g(\mathbf{K}, \mathbf{M}, \mathbf{N}) \\ & - \ g(\mathbf{K} - \mathbf{A}\mathbf{e}_j, \mathbf{M} - \mathbf{e}_j, \mathbf{N} - \mathbf{A}\mathbf{e}_j). \qquad (4.3) \end{aligned}$$

Let

$$H(\mathbf{z}, \mathbf{M}, \mathbf{N}) = \sum_{K_1=0}^{\infty} \cdots \sum_{K_p=0}^{\infty} h(\mathbf{K}, \mathbf{M}, \mathbf{N}) \prod_{i=1}^{p} z_i^{K_i}. \qquad (4.4)$$

To get a closed-form expression for $H(\mathbf{z}, \mathbf{M}, \mathbf{N})$, define $g(\mathbf{K}, \mathbf{M}, \mathbf{N})$ to be 0 whenever $K_i < 0$ for any i in the range $1 \le i \le p$. It can be shown that

$$\begin{aligned} H(\mathbf{z}, \mathbf{M}, \mathbf{N}) = {} & \prod_{i=1}^{p} (1 - z_i)^{-1} \prod_{\substack{k=1 \\ k \ne j}}^{r} G_k(\mathbf{z}, M_k, N_k) \\ & \times \left[G_j(\mathbf{z}, M_j, N_j) - \prod_{i=1}^{p} z_i^{a_{ij}} G_j(\mathbf{z}, M_j - 1, N_j - b_j) \right] (4.5) \end{aligned}$$

where $G_j(\mathbf{z}, M_j, N_j)$ is as in (3.14). The numerical difficulty disappears if, instead of computing $h(\mathbf{K}, \mathbf{M}, \mathbf{N})$ via (4.3), we compute it by transform inversion of the generating function expression (4.5). The inversion procedure and scaling are identical to those used for the standard generating function $G(\mathbf{z}, \mathbf{M}, \mathbf{N})$.

As noted above, if class-j jobs arrive in a Poisson process, then (4.1) also yields the call blocking, but not more generally. However, the call blocking always can be obtained by calculating the time blocking in a modified model. Let B_j be the class-j blocking probability (call blocking).

For notational simplicity only, consider the CS policy. Let $\mathbf{A} \equiv (a_{ij})$ be the requirements matrix. Then

$$\begin{aligned} B_j &= 1 - \frac{\sum_{\mathbf{n}: \mathbf{A}\mathbf{n} \le \mathbf{K} - \mathbf{a}_j} \lambda_j(n_j) \pi(\mathbf{n})}{\sum_{\mathbf{n}: \mathbf{A}\mathbf{n} \le \mathbf{K}} \lambda_j(n_j) \pi(\mathbf{n})} \\ &= 1 - \frac{\sum_{\mathbf{n}: \mathbf{A}\mathbf{n} \le \mathbf{K} - \mathbf{a}_j} \lambda_j(n_j) f(\mathbf{n})}{\sum_{\mathbf{n}: \mathbf{A}\mathbf{n} \le \mathbf{K}} \lambda_j(n_j) f(\mathbf{n})} \qquad (4.6) \end{aligned}$$

for $f(\mathbf{n})$ in (2.3). However, we can rewrite $\lambda_j(n_j) f(\mathbf{n})$ as $\lambda_j(0) \overline{f}(\mathbf{n})$ and thus (4.6) as

$$B_j = 1 - \frac{\sum_{\mathbf{n}: \mathbf{A}\mathbf{n} \le \mathbf{K} - \mathbf{a}_j} \overline{f}(\mathbf{n})}{\sum_{\mathbf{n}: \mathbf{A}\mathbf{n} \le \mathbf{K}} \overline{f}(\mathbf{n})} = 1 - \frac{\overline{g}(\mathbf{K} - \mathbf{a}_j)}{\overline{g}(\mathbf{K})}, \qquad (4.7)$$

where $\overline{f}(\mathbf{n})$ is the analog of $f(\mathbf{n})$ with $\lambda_j(m)$ replaced by $\overline{\lambda}_j(m) \equiv \lambda_j(m+1)$, and $\overline{g}(\mathbf{K})$ is the analog of $g(\mathbf{K})$ with $f(\mathbf{n})$ replaced by $\overline{f}(\mathbf{n})$. The same argument clearly holds for non-CS policies. We summarize this result for the combined UL/GM policy as follows.

Theorem 4.1 *With the combined UL/GM policy, the class-j blocking probability B_j coincides with the time-blocking quantities B_j^t in (4.1) for the modified model in which the class-j arrival-rate function is changed from $\lambda_j(m)$ to $\overline{\lambda}_j(m) \equiv \lambda_j(m+1)$.*

For the special case in which $\lambda_j(m) = \alpha_j + \beta_j m$,

$$\begin{aligned} \overline{\lambda}_j(m) &= \lambda_j(m+1) = \alpha_j + \beta_j(m+1) \\ &= (\alpha_j + \beta_j) + \beta_j m, \qquad (4.8) \end{aligned}$$

so that the modified model is a model of the same general form. For the BPP model, this approach to computing call blocking was pointed out by Dziong and Roberts [15], p. 273. Note that $\overline{\lambda}_j$ coincides with λ_j when there are Poisson arrivals and $\overline{\lambda}_j$ reduces to the arrival rate with one less class-j source when class j has a finite source input, agreeing with known properties. Van de Vlag and Awater [32] have recently developed an efficient procedure for computing call blocking probabilities for many classes. However, both of these papers only consider the CS policy.

5 Other Performance Measures

Let X_j represent the steady-state number of class-j jobs in service at an arbitrary time. The primary performance measures other than the probability of blocking are the marginal distribution $P(X_j \le l)$ and the k^{th} factorial moment, defined as

$$E[X_j^{(k)}] = E[X_j(X_j - 1) \cdots (X_j - k + 1)] \quad \text{for} \quad k \ge 1.$$

From (2.2)–(2.5), it is easy to see that these quantities are given by

$$P(X_j \le l) = \frac{g_d^{(j,l)}(\mathbf{K})}{g(\mathbf{K})} \qquad (5.1)$$

and

$$E[X_j^{(k)}] = \frac{g_m^{(j,k)}(\mathbf{K})}{g(\mathbf{K})}, \tag{5.2}$$

where

$$g_d^{(j,l)}(\mathbf{K}) = \sum_{\substack{\mathbf{n}\in S_p(\mathbf{K})\\ n_j\le l_j}} f(\mathbf{n}) \tag{5.3}$$

and

$$g_m^{(j,k)}(\mathbf{K}) = \sum_{\mathbf{n}\in S_p(\mathbf{K})} n_j(n_j-1)\cdots(n_j-k+1)f(\mathbf{n}). \tag{5.4}$$

Note that the performance measures in (5.1) and (5.2) are ratios of a modified normalization constant and the standard normalization constant. We show that it is possible to construct the generating function of each type of modified normalization constant, and that it has an expression similar to $G(\mathbf{z})$, so that it may be directly inverted. Hence, each performance measure requires the inversion of just two generating functions. Specifically, let $G_d^{(j,l)}(\mathbf{z})$ and $G_m^{(j,k)}(\mathbf{z})$ represent the generating functions of $g_d^{(j,l)}(\mathbf{K})$ and $g_m^{(j,k)}(\mathbf{K})$, respectively. We can get expressions in the CS case if we work with (2.8) and the UG (combined UL/GM) case if we work with (3.10). To save space, we only consider the UG case. As in Section 3, we include $\mathbf{M}$ and $\mathbf{N}$ as arguments of normalization constants and their generating functions. (Note that the CS case is a special case of the UG case with $N_j = 0$ and $M_j = \infty$, but it does not have the restriction $a_{ij} = \delta_{ij}b_j$).

It is easy to see that

$$G_d^{(j,l)}(\mathbf{z},\mathbf{M},\mathbf{N}) = \prod_{i=1}^{p}(1-z_i)^{-1}G_{dj}^{(l)}(\mathbf{z},M_j,N_j)\prod_{\substack{q=1\\ q\ne j}}^{r}G_q(\mathbf{z},M_q,N_q) \tag{5.5}$$

and

$$G_m^{(j,k)}(\mathbf{z},\mathbf{M},\mathbf{N}) = \prod_{i=1}^{p}(1-z_i)^{-1}G_{mj}^{(k)}(\mathbf{z},M_j,N_j)\prod_{\substack{q=1\\ q\ne j}}^{r}G_q(\mathbf{z},M_q,N_q), \tag{5.6}$$

where $G_q(\mathbf{z},M_q,N_q)$ is as in (3.10),

$$G_{dj}^{(l)}(\mathbf{z},M_j,N_j) = y_j^{N_j/b_j}\sum_{n_j=0}^{l} f_j(n_j) \quad \text{for } l \le \lfloor N_j/b_j\rfloor, \text{ and}$$

$$= y_j^{N_j/b_j}\sum_{n_j=0}^{\lfloor N_j/b_j\rfloor} f_j(n_j) + \sum_{n_j=\lfloor N_j/b_j\rfloor+1}^{l} f_j(n_j)y_j^{n_j} \tag{5.7}$$

for $M_j \ge l > \lfloor N_j/b_j\rfloor$,

$$G_{mj}^{(k)}(\mathbf{z},M_j,N_j)$$

$$= y_j^{N_j/b_j}\sum_{n_j=k}^{\lfloor N_j/b_j\rfloor} f_j(n_j)n_j(n_j-1)\cdots(n_j-k+1)$$

$$+ \sum_{n_j=\lfloor N_j/b_j\rfloor+1}^{m_j} f_j(n_j)n_j(n_j-1)\cdots(n_j-k+1)y_j^{n_j}$$

for $k \le \lfloor N_j/b_j\rfloor$, and

$$= \sum_{n_j=k}^{M_j} f_j(n_j)n_j(n_j-1)\cdots(n_j-k+1)y_j^{n_j} \tag{5.8}$$

for $M_j \ge k > \lfloor N_j/b_j\rfloor$, and

$$y_j = \prod_{i=1}^{p} z_i^{\delta_{ij}b_j} = \prod_{i=1}^{p} z_i^{a_{ij}}. \tag{5.9}$$

Note that $G_d^{(j,l)}(\mathbf{z},\mathbf{M},\mathbf{N})$ and $G_m^{(j,k)}(\mathbf{z},\mathbf{M},\mathbf{N})$ are just as easy to compute and then invert (see Section 8) as $G(\mathbf{z},\mathbf{M},\mathbf{N})$. Also, typically, whenever there is a simple closed-form expression for $G(\mathbf{z},\mathbf{M},\mathbf{N})$ (as for the unbuffered CS policy with BPP arrivals in (2.13) or the buffered CS/UG policy with Poisson arrivals in (3.14))) there will be a similar one for $G_d^{(j,l)}(\mathbf{z},\mathbf{M},\mathbf{N})$ and $G_m^{(j,k)}(\mathbf{z},\mathbf{M},\mathbf{N})$ as well. For the unbuffered model with UG policy and BPP arrivals, using the expression for $f_j(n_j)$ from (2.12), it is easy to show that

$$E[X_j^{(k)}] = \frac{\alpha_j(\alpha_j+\beta_j)\cdots(\alpha_j+(k-1)\beta_j)}{\mu_j^k} \times \frac{\overline{g}(\mathbf{K}-k\mathbf{A}\mathbf{e}_j,\ \mathbf{M}-k\mathbf{e}_j,\ \mathbf{N}-k\mathbf{A}\mathbf{e}_j)}{g(\mathbf{K},\mathbf{M},\mathbf{N})}, \tag{5.10}$$

where $\overline{g}$ in the numerator implies that we have to consider a system with α_j replaced by $\alpha_j + k\beta_j$.

6 Modeling and Preliminary Analysis for the Unbuffered Variant with BPP Arrivals

In Section 4 we assumed that the sources really are of type BPP, e.g., because they are in finite-source models or in models with controlled arrival rates. However, another important use of the BPP model is represent non-Poisson traffic, which occurs in overflow traffic associated with alternative routing. Non-Poisson traffic can be characterized approximately via a *peakedness* parameter [13, 14, 15, 37]. Peakedness is defined as the ratio of the variance to the mean of the number of jobs in service in the associated infinite-capacity system. For Poisson arrivals, the steady-state distribution in the infinite-capacity system is Poisson, so that the peakedness is 1. For more bursty arrival processes, the peakedness is greater than 1; for less bursty arrival processes, the peakedness is less than 1.

A way to approximately represent non-Poisson traffic in our product-form model is to approximate the actual arrival process by a BPP arrival process with the same arrival rate and peakedness. For an unbuffered model with

$\mu_j(k) = k\mu_j$ and BPP arrival processes with state-dependent rates $\lambda_j(k) = \alpha_j + k\beta_j$, the means and variances in the infinite-capacity system are

$$m_j = \alpha_j/(\mu_j - \beta_j) \quad \text{and} \quad v_j = \mu_j\alpha_j/(\mu_j - \beta_j)^2. \quad (6.1)$$

From (6.1), we see that the two BPP parameters for each class can be expressed as

$$\alpha_j = m_j\mu_j/z_j \quad \text{and} \quad \beta_j = \mu_j(z_j - 1)/z_j \quad (6.2)$$

where $z_j \equiv v_j/m_j$ is the peakedness. (This traditional peakedness notation overlaps with our convention for complex variables, but the context makes it clear which is intended.)

Having obtained α_j and β_j, we wish to compute the blocking probabilities. This could be done directly by applying Section 4, but as noted by Delbrouck [13], a better approximation is obtained if we calculate the blocking probability indirectly via the mean number of active jobs in the finite-capacity system with the BPP arrival process. Hence, the next step is to compute the mean number of class-j jobs in service in the actual system with capacity constraints, which we denote by $\tilde{m}_j$. Note that Delbrouck [13] computed $\tilde{m}_j$ in a much simpler system with single rate, single resource and CS policy. However, we have shown in Section 5 (equation (5.10)) that simple expressions for $\tilde{m}_j$ exist even in our much more general model. Specifically, with combined UL/GM policy,

$$\begin{aligned}\tilde{m}_j &= E[X_j^{(1)}] \\ &= \frac{\alpha_j \overline{g}(\mathbf{K} - \mathbf{A}\mathbf{e}_j,\ \mathbf{M} - \mathbf{e}_j,\ \mathbf{N} - \mathbf{A}\mathbf{e}_j)}{\mu_j g(\mathbf{K}, \mathbf{M}, \mathbf{N})}, \quad (6.3)\end{aligned}$$

where the symbol $\overline{g}$ in the numerator of (6.3) indicates that we have to consider a system with α_j replaced by $\alpha_j + \beta_j$. Note that this replacement is only done in the numerator. Finally, the expression for call blocking is

$$\begin{aligned}B_j &= 1 - \frac{\tilde{m}_j}{m_j} = 1 - \left(1 - \frac{\beta_j}{\mu_j}\right) \\ &\quad \times \frac{\overline{g}(\mathbf{K} - \mathbf{A}\mathbf{e}_j,\ \mathbf{M} - \mathbf{e}_j,\ \mathbf{N} - \mathbf{A}\mathbf{e}_j)}{g(\mathbf{K}, \mathbf{M}, \mathbf{N})}. \quad (6.4)\end{aligned}$$

7 Eliminating Very Lightly Loaded Resources

We now show how reduce the dimension of the generating function before performing the inversion by eliminating very lightly loaded resources. This step is done before doing conditional decomposition in Section 3.1 of [6].

For this purpose, we can use the infinite-server means m_j and variances v_j in (6.1) to estimate the load on each resource. If there were no capacity constraints, then the mean and variance of the number of resource units in use on resource i would be

$$M_i = \sum_{j=1}^{r} m_j a_{ij} \quad \text{and} \quad V_i = \sum_{j=1}^{r} v_j a_{ij}^2 \quad (7.1)$$

for $1 \le i \le p$. We call M_i the *resource-i offered load* and V_i the *resource-i variance.*

The resource-i offered load and variance are very important for recognizing when the analysis of large multi-resource problems can be simplified before doing any computation. If M_i is much smaller than K_i, then we can simply ignore the constraint imposed by resource i. This reduces the dimension of the generating function.

We can better estimate the importance of the constraint imposed by resource i by approximating the distribution of the number of required circuits by a normal distribution with mean M_i and variance V_i in (7.1). The normal distribution tends to be appropriate because, without capacity constraints, the resource-i occupancy is the convolution of the r occupancy distributions for each class. The normal approximation will tend to be more accurate when r is large, by virtue of the central limit theorem.

Let $N(m, \sigma^2)$ represent a normally distributed random variable with mean m and variance σ^2. Let $\phi(x)$ and $\Phi(x)$ be the density and cdf of $N(0,1)$, respectively; i.e., $\phi(x) = (2\pi)^{-1/2}\exp(-x^2/2)$ and $\Phi(x) = P(N(0,1) \le x)$. Let $\Phi^c(x) = 1 - \Phi(x)$. Let X_i be the steady-state number of occupied resource units in resource i. The simple normal approximation without capacity constraints has X_i approximately distributed as $N(M_i, V_i)$ for M_i and V_i in (7.1). This means that $(X_i - M_i)/\sqrt{V_i}$ is distributed approximately as $N(0,1)$.

Thus, to approximately quantify the importance of the various resource constraints, we can solve the equations

$$M_i + \gamma_i\sqrt{V_i} = K_i, \quad (7.2)$$

where γ_i is called the *resource-i binding parameter.* Resources with *larger* binding parameters γ_i values will tend to be *less binding*, and thus less important. If γ_i is suitably large, then we can delete resource i from the model before performing the computation. For previous work on normal approximations, see [14, 35] and references there. We are now suggesting these approximations as an initial step before applying our algorithm.

We have calculated the means and variances in this section under the simplifying assumption that there are no capacity constraints. However, when we use the UL and GM bounds, the individual class means and variances m_j and v_j may change dramatically. For instance, if the GM bound is significantly bigger than m_j, then the new mean is approximately the GM bound itself and the new variance is approximately 0. More generally, we can calculate approximate adjusted means and variances for each class using properties of *conditioned* normal distributions, as we now show.

We assume that the occupancy for the class in question is approximately distributed as the conditional normal random variable $(N(m, \sigma^2) \mid N(m, \sigma^2) \le U)$ where U is the upper limit parameter. (We omit the j subscript.) When we consider the number of resource units used, we need to multiply m by b and σ^2 by b^2, where $b = a_{ij}$ for resource i and class j.

The key to our analysis is the following property of conditioned normal distributions.

Lemma 7.1 *For $-\infty < L < U \leq \infty$,*

$$E[N(m,\sigma^2) \mid L \leq N(m,\sigma^2) \leq U]$$
$$= m + \sigma \frac{\phi((L-m)/\sigma) - \phi((U-m)/\sigma)}{\Phi((U-m)/\sigma) - \Phi((L-m)/\sigma)} \qquad (7.3)$$

and

$$E[N(m,\sigma^2)^2 \mid L \leq N(m,\sigma^2) \leq U]$$
$$= m^2 + \sigma^2 + 2m\sigma \left(\frac{\phi((L-m)/\sigma) - \phi((U-m)/\sigma))}{\Phi((U-m)/\sigma) - \Phi((L-m)/\sigma)} \right)$$
$$+ \sigma \frac{((L-m)\phi((L-m)/\sigma) - (U-m)\phi((U-m)/\sigma))}{\Phi((U-m)/\sigma) - \Phi((L-m)/\sigma))}. \qquad (7.4)$$

Proof. For the mean, note that $x\phi(x) = -\phi'(x)$ for all x, so that

$$E[N(0,1) \mid L \leq N(0,1) \leq U] = \frac{\phi(L) - \phi(U)}{\Phi(U) - \Phi(L)}$$

from which (7.3) follows easily. Similarly, for the second moment, note that $x^2\phi(x) = \phi(x) + \phi''(x)$, so that

$$E[N(0,1)^2 \mid L \leq N(0,1) \leq U] = 1 + \frac{L\phi(L) - U\phi(U)}{\Phi(U) - \Phi(L)},$$

from which (7.4) follows easily. ■

We now apply Lemma 7.1 to calculate the (approximate) first two moments of the capacity used by a single class, in the presence of guaranteed minimum and an upper limit parameters.

Theorem 7.1 *Let L and U be guaranteed minimum and an upper limit parameters. Assuming that the occupancy for some customer can be approximated by the conditional normal variable $(N(m,\sigma^2) \mid N(m,\sigma^2) \leq U)$, the first two moments of the capacity used at any time are*

$$EC(L,U) = bLP(N(m,\sigma^2) \leq L \mid N(m,\sigma^2) \leq U)$$
$$+ \big[bE(N(m,\sigma^2) \mid L \leq N(m,\sigma^2) \leq U)$$
$$\times P(L \leq N(m,\sigma^2) \mid N(m,\sigma^2) \leq U)\big]$$
$$= bL \frac{\Phi((L-m)/\sigma)}{\Phi((U-m)/\sigma)}$$
$$+ bX \left(\frac{\Phi((U-m)/\sigma) - \Phi((L-m)/\sigma)}{\Phi((U-m)/\sigma)} \right), \qquad (7.5)$$

where

$$X = m + \sigma \frac{\phi((L-m)/\sigma) - \phi((U-m)/\sigma)}{\Phi((U-m)/\sigma) - \Phi((L-m)/\sigma)} \qquad (7.6)$$

and

$$E[C(L,U)^2] = b^2L^2P(N(m,\sigma^2) \leq L \mid N(m,\sigma^2) \leq U)$$
$$+ \big[b^2E[N(m,\sigma^2)^2 \mid L \leq N(m,\sigma^2) \leq U)$$
$$\times P(L \leq N(m,\sigma^2) \mid N(m,\sigma^2) \leq U)\big]$$
$$= b^2L^2 \frac{\Phi((L-m)/\sigma)}{\Phi((U-m)/\sigma)}$$
$$+ b^2Y \left(\frac{\Phi((U-m)/\sigma) - \Phi((L-m)/\sigma)}{\Phi((U-m)/\sigma)} \right), \qquad (7.7)$$

where

$$Y = m^2 + \sigma^2$$
$$+ 2m\sigma \left(\frac{\phi((L-m)/\sigma) - \Phi((U-m)/\sigma)}{\Phi((U-m)/\sigma) - \Phi((L-m)/\sigma)} \right)$$
$$+ \sigma \left(\frac{(L-m)\phi((L-m)/\sigma) - (U-m)\phi((U-m)/\sigma)}{\Phi((U-m)/\sigma) - \Phi((L-m)/\sigma)} \right). \qquad (7.8)$$

If, in addition, $\Phi((U-m)/\sigma) \approx 1$ and $\phi((U-m)/\sigma) \approx 0$, then

$$EC(L,U) \approx bm - b(m-L)\Phi((L-m)/\sigma)$$
$$+ b\sigma\phi((L-m)/\sigma) \qquad (7.9)$$

and

$$E[C(L,U)^2] \approx b^2L^2\Phi((L-m)/\sigma)$$
$$+ b^2(m^2+\sigma^2)\Phi^c((L-m)/\sigma)$$
$$+ b^2[2m\sigma + (L-m)]\phi((L-m)/\sigma)). \qquad (7.10)$$

Note that when we guarantee the average rate, i.e., when $L = m$, (7.9) reduces to

$$EC(L,U) = mb + \sigma b\phi(0)$$
$$= mb + \frac{\sigma b}{\sqrt{2\pi}} \approx (m + 0.4\sigma)b, \qquad (7.11)$$

while the variance is

$$\operatorname{Var} C(L,U) = \sigma^2 b^2(\pi - 1)/2\pi \approx 0.34\sigma^2 b^2. \qquad (7.12)$$

In summary, the idea is to apply Theorem 7.1 to compute the approximate mean and variance of the capacity used by each *class* j. Next we add these means and variances, as in (7.1), to compute the mean and variance of the capacity needed for each *resource* i. We then apply (7.2) to identify resources that can be deleted from the model. We evaluate the binding parameters using the standard normal tail probabilities $\Phi^c(\gamma_i)$. (Recall that $\Phi^c(2) = 0.023$, $\Phi^c(3) = 0.0014$, $\Phi^c(4) = 0.000032$ and $\Phi^c(5) \approx 3 \times 10^{-7}$.) Hence, we can often remove resource i if $\gamma_i \geq 5$. We should also make adjustments for requirements of multiple resource units. The blocking probability for a class that requires b resource units is roughly b times the blocking probability of a class that requires only 1 resource unit.

Finally, with the resulting reduced model, we can apply dimension reduction using conditional decomposition to further reduce the required computation [6, 8].

8 Scaling in the Inversion Algorithm

Most of the algorithm is as in [6, 8], so we will be brief. Given a p-dimensional generating function $G(\mathbf{z})$, we first do the dimension reduction to determine the order in which the variables should be inverted. We then perform (up to) p one-dimensional inversions recursively, using the algorithm in [9] which is based on the Fourier-series method.

An important component of the algorithm is an appropriate scaling of the generating function in each step of the inversion. The primary purpose of the scaling is to effectively control the aliasing error of the inversion procedure. (This is done in conjunction with geometric damping as described in [6] and [9]). The scaling also effectively avoids the floating point exception problem commonly encountered in computing very large or small normalization constants.

Let the innermost level of inversion be with respect to z_1 and successive outer levels of inversion be done with respect to $z_2, z_3, \ldots, z_p$. At the j^{th} level, the inversion is with respect to z_j and let the generating function involved at this stage be defined as $G^{(j)}(z_j)$. Instead of inverting it as is, we invert the scaled generating function

$$\overline{G}^{(j)}(z_j) = \alpha_{0j}\overline{G}^{(j)}(\alpha_j z_j). \tag{8.1}$$

Our main concern is to find an appropriate $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_p)$ and $\boldsymbol{\alpha}_0 = (\alpha_{01}, \alpha_{02}, \ldots, \alpha_{0p})$. In [6] we did this for Poisson arrivals and unbuffered resource-sharing models. We develop the following unified heuristic scaling algorithm to treat all the generating functions in the paper. All generating functions (including the modified ones considered in Section 5) have the generic form

$$G(\mathbf{z}) = \prod_{i=1}^{p}(1 - z_i)^{-1}\prod_{j=1}^{r} G_j(\mathbf{z}). \tag{8.2}$$

The scale vector $\boldsymbol{\alpha}$ should be a maximal vector satisfying $0 < \alpha_i \leq 1$ for $i = 1, 2, \ldots, p$ and

$$\sum_{j=1}^{r}\left(\prod_{k=1}^{i-1} r_k^{a_{kj}}\right) z_i \frac{\partial}{\partial z_i}\ln G_j(\mathbf{z})\bigg|_{\mathbf{z}=\boldsymbol{\alpha}} \leq K_i \tag{8.3}$$

for $i = 1, 2, \ldots, p$, where

$$r_k = 10^{-(\gamma_k/2l_k K_k)}$$

as in Section 3.2 of [6]. "Maximal" means that (8.3) should be satisfied with equality for at least one i unless $\boldsymbol{\alpha} = (1, 1, \ldots, 1)$. The remaining scale parameters α_{0i}, $1 \leq i \leq p$, are obtained recursively starting with $i = p$ by

$$\prod_{k=1}^{p}\alpha_{0k} = \left[\prod_{j=1}^{r} G_j(\alpha_1 r_1, \alpha_2 r_2, \ldots, \alpha_{i-1}r_{i-1}, \alpha_i, \alpha_{i+1}, \ldots, \alpha_p)\right]^{-1}. \tag{8.4}$$

The scaling, although heuristic, has been tested extensively over a wide range of parameter values for all generating functions in this paper and elsewhere (e.g., [8]) and in all cases it performed well. Also, in the special case of Poisson arrivals and unbuffered resource-sharing model, the scaling becomes identical to that developed in [6].

Recently it has been shown [4] that there is an asymptotic justification of the heuristic scaling in the one-dimensional case based on the theory of the saddle point method. Since the value of integrand falls off very fast away from the saddle point, this theory also gives further justification for why we can have great computational saving by judicious truncation in the inversion formula.

9 Multiplicities

If two or more classes have the same parameters (traffic parameters, resource requirements, UL and GM parameters), then we say that there is a *multiplicity.* We can exploit multiplicities to significantly reduce the required computation.

Let $\overline{r}$ be the number of different *types* of job classes and let the j^{th} type have multiplicity m_j. Then the total number of job classes is

$$r = \sum_{j=1}^{\overline{r}} m_j. \tag{9.1}$$

If the generating function of interest can be written as

$$G(\mathbf{z}) = \prod_{i=1}^{p}(1 - z_i)^{-1}\prod_{j=1}^{r} G_j(\mathbf{z}), \tag{9.2}$$

then it can be rewritten as

$$G(\mathbf{z}) = \prod_{i=1}^{p}(1 - z_i)^{-1}\prod_{j=1}^{\overline{r}} G_j(\mathbf{z})^{m_j}. \tag{9.3}$$

The computational complexity in evaluating (9.2) is $O(r)$, while the computational complexity in evaluating (9.3) is $O(\overline{r})$.

For the unbuffered variant with Poisson arrivals and the CS policy, we can obviously replace all classes of type j by a single class with traffic intensity $m_j\rho_j$ and hence the benefit of multiplicity can be trivially achieved by any algorithm. However, in other cases it is not straightforward for other algorithms (e.g., recursive algorithms) to benefit from multiplicities. In some cases a general multiplicity is not even allowed in other algorithms. For example, Kamoun and Kleinrock [16] allow either $\overline{r} = 1$ or $m_j = 1$ for all j in their algorithm for the buffered variant.

10 Computational Complexity

We now roughly analyze the computational complexity of the inversion algorithm. For simplicity, assume that the capacity of each resource is K. Let C_P represent the computational complexity for sharing policy P, where P may be CS, UL, GM or UG (combined UL/GM). For the inversion algorithm, the computational complexity is the same for state-dependent inputs as for Poisson arrivals, so we do not focus on that aspect. The main computational burden is carrying out the p-fold nested inversion in (3.3)–(3.5) of [6]. Other work, such as finding the scale parameters is insignificant compared to that. A straightforward application of our algorithm in the CS case to compute one normalization constant would require $O(K^p)$ evaluations of the generating function, each of which would involve $O(r)$ work. In order to compute the blocking probability for each class, we need to compute $r+1$ or $2r$ normalization constants, but in Section 5.2 of [6] we have shown that all this work can be done in time $O(1)$ by sharing the bulk of the computation (requiring storage only of $O(r)$). Without further enhancements, this yields $C_{CS} = O(rK^p)$.

However, in Section 5.1 of [6] we have shown that we can use truncation to reduce K to $\overline{K} = O(\sqrt{K})$ and, with special structure (see Section 3.1 of [6]), we can reduce p to $\overline{p} \ll p$. Moreover, with multiplicities, we can reduce r to $\overline{r}$. So, finally, we get

$$C_{CS} = O(\overline{r}\,\overline{K}^{\overline{p}}), \tag{10.1}$$

where $\overline{K} \le K$, $\overline{p} \le p$, $\overline{r} \le r$, and

$$\begin{aligned} &\overline{K} = O(\sqrt{K}) \quad \text{for large } K, \\ &\overline{p} \ll p \quad \text{with special structure, and} \\ &\overline{r} \ll r \quad \text{for large } r \text{ and large multiplicities.} \end{aligned} \tag{10.2}$$

By contrast, for the Delbrouck [14] recursion for a single resource ($p = 1$),

$$C_{CS} = O(rK^2), \tag{10.3}$$

and so we are much faster even for a moderate K. Recently van de Vlag and Awater [32] improved the Delbrouck recursion to have a computational complexity $O(rK)$. Their improvement also extends to multiple resources in which case they get

$$C_{CS} = O(rK^p). \tag{10.4}$$

Comparing (10.1) and (10.4), we see that we are faster when we can replace K, p or r by $\overline{K}$, $\overline{p}$ or $\overline{r}$ in (10.2), respectively.

For general state-dependent arrival and service rates, we compute using (2.9). Assuming $K_i = K$ for all i and $a_{ij} = a$ for all i and j, we get the computational complexity

$$C_{CS} = O((K/a)\overline{r}\,\overline{K}^{\overline{p}}) \tag{10.5}$$

for $\overline{K}, \overline{p}$ and $\overline{r}$ in (10.2). For the combined UL and GM policy (UG), the computational complexity is

$$C_{UG} = O(M\overline{r}\,\overline{K}^{\overline{p}}), \tag{10.6}$$

where we have assumed the class upper limit $M_j = M$ for all j. For the case of the buffered variant with Poisson arrivals, $C_{UG} = C_{CS}$; see (3.14).

For state-dependent rates or with UL/GM policies, there are no previous algorithms to compare with except for Kamoun and Kleinrock [16] which applies only to a special case. We do not do any elaborate comparison with [16], but it appears that with UL/GM policy and many classes, our algorithm would be much faster than theirs. In the next section we develop a new convolution-based recursion to treat the most general model in this paper. Its computational complexity is

$$C_{CON} = O(rK^2p). \tag{10.7}$$

Comparing (10.7) with (10.5) and (10.6) we see that our inversion algorithm is much faster than the convolution algorithm.

11 Recursive Algorithms from the Generating Functions

In this section we show that the generating functions we have derived can be used to derive recursions for the normalization constants. This method for obtaining recursions was first proposed by Reiser and Kobayashi [29] for closed queueing networks, but it has not been widely used since; e.g., the method is not used by Kaufman [17], Roberts [30] or Dziong and Roberts [15] to derive their recursions. However, generating functions have been used to derive recursions for loss models by Delbrouck [14], Mitra and Morrison [26] and Morrison [27], Appendix A.

As noted in [29], a starting point for the recursions is the fact that the coefficients of a generating function that is a product of two generating functions is the convolution of the coefficients from the two component generating functions. This remains true with vectors using multidimensional convolution. To express the result, let

$$G_\eta(\mathbf{z}) = \sum_{K_1=0}^{\infty} \cdots \sum_{K_p=0}^{\infty} g_\eta(\mathbf{K}) z_1^{K_1} \cdots z_p^{K_p}\ , \tag{11.1}$$

where $\mathbf{K} = (K_1, \ldots, K_p)$, $\mathbf{z} = (z_1, \ldots, z_p)$ and η is an index, which may be an integer or a vector of integers.

Lemma 11.1 *If $G_{1,2}(\mathbf{z}) = G_1(\mathbf{z})G_2(\mathbf{z})$ for generating functions defined in (11.1), then*

$$g_{1,2}(\mathbf{K}) = \sum_{k_1=0}^{K_1} \cdots \sum_{k_p=0}^{K_p} g_1(\mathbf{k}) g_2(\mathbf{K} - \mathbf{k}), \tag{11.2}$$

where $\mathbf{k} = (k_1, \ldots, k_p)$.

It is straightforward to apply Lemma 11.1 repeatedly to our generating functions in (2.7) because they are expressed as products of $r+1$ generating functions. (Note that $\prod_{i=1}^{p}(1-z_i)^{-1}$ can be regarded as a single factor.) However, in general the computational complexity is quite high. In particular, for each value of $\mathbf{K}$, $O(\prod_{i=1}^{p} K_i)$ operations are needed. However, since we need repeat this operation, we need to compute $g_{1,2}(\mathbf{K})$ for the $O(\prod_{i=1}^{p} K_i)$ possible values of $\mathbf{K}$. Thus, the computational complexity of (11.1) is really $O(\prod_{i=1}^{p} K_i^2)$, assuming that each factor can be inverted separately.

We can also exploit the special structure of our generating functions. For example, we have the factor $\prod_{i=1}^{p}(1-z_i)^{-1}$ in each case.

Lemma 11.2 *If the condition of Lemma 11.1 holds with* $G_2(\mathbf{z}) = \prod_{i=1}^{p}(1-z_i)^{-1}$, *then*

$$g_{1,2}(\mathbf{K}) = \sum_{k_1=0}^{K_1} \cdots \sum_{k_p=0}^{K_p} g_1(\mathbf{k}). \tag{11.3}$$

for $K_i = k_i$, *so that (using the method of inclusion and exclusion)*

$$\begin{aligned} g_{1,2}(\mathbf{K}) &= g_1(\mathbf{K}) \\ &\quad + \sum_{i=1}^{p} g_{1,2}(\mathbf{K}-\mathbf{e}_i) - \sum_{i=1}^{p} \sum_{\substack{j=1 \\ i\neq j}}^{p} g_{1,2}(\mathbf{K}-\mathbf{e}_i-\mathbf{e}_j) \\ &\quad + \sum_{\substack{i=1 \\ i\neq j,}}^{p} \sum_{\substack{j=1 \\ i\neq k,}}^{p} \sum_{\substack{k=1 \\ j\neq k}}^{p} g_{1,2}(\mathbf{K}-\mathbf{e}_i-\mathbf{e}_j-\mathbf{e}_k) \\ &\quad - \cdots \pm g_{1,2}\left(\mathbf{K} - \sum_{i=1}^{p} \mathbf{e}_i\right). \end{aligned} \tag{11.4}$$

Proof. For (11.3), apply Lemma 11.1 noting that $g_2(\mathbf{k}) = 1$ for all $\mathbf{k}$. ∎

Formula (11.4) does not appear to be especially helpful for computation, because it involves subtraction of large, nearly equal quantities.

The computation required for (11.3) is $O(\prod_{i=1}^{p} K_i)$ if done for only one value of $\mathbf{K}$. This is possible if (11.3) is applied only in the last step. From (2.7), we see that we use (11.3) once and (11.2) $r-1$ times. If we use (11.3) in the last step, then the computational complexity of (11.3) is dominated by that of (11.2). Hence the overall computational complexity is $O((r-1)\prod_{i=1}^{p} K_i^2) \approx O(rK^{2p})$ for $K_i \approx K$, $1 \le i \le p$.

In order to determine the components $g_j(\mathbf{k})$ needed in the convolution algorithm, we apply (2.9). Given (2.9) and

$$G_j(\mathbf{z}) = \sum_{k_1=0}^{\infty} \cdots \sum_{k_p=0}^{\infty} g_j(\mathbf{k}) \prod_{i=1}^{p} z_i^{k_i}, \tag{11.5}$$

we obtain

$$g_j(la_{1j}, ..., la_{pj}) = f_j(l) \tag{11.6}$$

for $l = 0, 1, \ldots, L_j$, where

$$L_j \equiv \min_i \{K_i / a_{ij}\} \tag{11.7}$$

and $g_j(\mathbf{k}) = 0$ for all other $\mathbf{k}$. Hence, we can compute $g_j(\mathbf{k})$ for each j and $\mathbf{k}$ using (11.6) and (2.3).

Note that the algorithm we have derived is very general. It allows arbitrary state-dependent arrival and service rates. We have just shown how to treat the CS policy using (2.9), but it also applies in essentially the same way to treat the combined UL and GM policy using (3.10) or (3.11). (We illustrate the convolution algorithm with the UL policy in the example of Section 14.)

It is to be noted that with special structure alternative faster recursive algorithms can be developed. In particular, the generalized Kaufman [17] – Roberts [30] recursion has complexity $O(rK^p)$. The same is true for the improved algorithm for the unbuffered variant with BPP arrivals by van de Vlag and Awater [32]. We now show how the generalized Kaufman-Roberts algorithm (the Poisson case of [14]) can be derived from the generating function.

Theorem 11.1 *If*

$$G(\mathbf{z}) = \exp\left[\sum_{j=1}^{r} \rho_j \prod_{i=1}^{p} z_i^{a_{ij}}\right], \tag{11.8}$$

for positive constants ρ_j *and nonnegative integers* a_{ij}, *then*

$$K_i g(\mathbf{K}) = \sum_{j=1}^{r} \rho_j a_{ij} g(\mathbf{K} - \mathbf{a}_j), \quad 1 \le i \le p, \tag{11.9}$$

where $\mathbf{a}_j = (a_{1j}, \ldots, a_{pj})$.

Proof. Let $G^{(i)}(\mathbf{z})$ be the partial derivative of $G(\mathbf{z})$ with respect to z_i. By taking the logarithm of (11.8) and then the partial derivative with respect to z_i, we get

$$\frac{G^{(i)}(\mathbf{z})}{G(\mathbf{z})} = \sum_{j=1}^{r} \rho_j a_{ij} \left(\prod_{i=1}^{p} z_i^{a_{ij}}\right) z_i^{-1}. \tag{11.10}$$

On the one hand,

$$G^{(i)}(\mathbf{z}) = \sum_{K_1=0}^{\infty} \cdots \sum_{K_p=0}^{\infty} K_i g(\mathbf{K}) z_1^{K_1} \cdots z_p^{K_p} z_i^{-1}, \tag{11.11}$$

while on the other hand, by (11.10),

$$\begin{aligned} G^{(i)}(\mathbf{z}) &= \sum_{j=1}^{r} \rho_j a_{ij} \left(\prod_{i=1}^{p} z_i^{a_{ij}}\right) z_i^{-1} G(\mathbf{z}) \\ &= \sum_{j=1}^{r} \rho_j a_{ij} \sum_{K_1=0}^{\infty} \cdots \sum_{K_p=0}^{\infty} \Big[g(\mathbf{K}) z_1^{K_1+a_{ij}} \\ &\quad \cdots z_i^{K_i+a_{ij}-1} \cdots z_p^{K_p+a_{pj}} \Big]. \end{aligned} \tag{11.12}$$

Matching coefficients of $(z_1^{K_1} \cdots z_i^{K_i-1} \cdots z_p^{K_p})$ in (11.11) and (11.12) yields (11.9). ∎

If we at first do (11.9) and then do (11.3) in the last step, then we have a required computation of $O(\prod_{i=1}^{p} K_i)+$

$O(r\prod_{i=1}^{p} K_i) = O((r+1)\prod_{i=1}^{p} K_i)$ and a storage requirement of $O(2\prod_{i=1}^{p} K_i)$.

We now treat the two non-Poisson BPP cases in Section 2.2. The following theorem applies to *each* of the r factors. The overall algorithm has computational complexity $O(rK^{2p})$, but it can effectively handle multiplicities.

Theorem 11.2 *If*

$$G(\mathbf{z}) = \left(1 + \rho \prod_{i=1}^{p} z_i^{a_i}\right)^c \tag{11.13}$$

for constants c and ρ, and nonnegative integers a_i, as in (2.13), then

$$g(\mathbf{K}) = \rho \left(\frac{ca_i - K_i + a_i}{K_i}\right) g(\mathbf{K} - \mathbf{a}) \tag{11.14}$$

if $a_i > 0$, $1 \le i \le p$, for $\mathbf{a} = (a_1, \ldots, a_p)$.

Proof. Differentiate the generating function with respect to z_i and get

$$G^i(\mathbf{z}) = c\left(1 + \rho\prod_{i=1}^{p} z_i^{a_i}\right)^{c-1} \rho a_i z_i^{-1} \prod_{k=1}^{p} z_k^{a_k},$$

so that

$$G^{(i)}(\mathbf{z})\left(1 + \rho\prod_{i=1}^{p} z_i^{a_i}\right) = c\rho G(\mathbf{z}) a_i z_i^{-1} \prod_{i=1}^{p} z_i^{a_i}. \tag{11.15}$$

Then identify the coefficients of $z_1^{K_1} \cdots z_i^{K_i - 1} \cdots z_p^{K_p}$ on both sides of (11.15), exploiting (11.11), to obtain

$$K_i g(\mathbf{K}) + \rho(K_i - a_i) g(\mathbf{K} - a) = c\rho a_i g(\mathbf{K} - \mathbf{a}) \tag{11.16}$$

from which (11.14) follows. ∎

As a simple check on (11.14), note that when $p = 1$ and $a_i = a_1 = 1$, (11.14) should reduce to the well known recursion for the binomial and negative binomial distributions, namely,

$$\frac{g(K)}{g(K-1)} = \rho\frac{(c - K + 1)}{K}. \tag{11.17}$$

12 The Kelly Example

We now give examples illustrating the numerical inversion algorithm. Two single-resource examples not discussed here appear in [7]. All computations were done on a SUN SPARC-2 workstation.

We first consider the Kelly example used in Sections 1 and 9 of [6], which originally came from [20]. It has five resources. The resources are five links connecting five nodes, as shown in Figure 1. (The nodes themselves play no role.)

Figure 1: The Kelly example.

In this example we consider the 6 routes $\{1\}$, $\{2\}$, $\{1,2\}$, $\{3,5\}$, $\{4,5\}$, $\{1,3,5\}$. The standard example has the CS policy, Poisson arrivals and single circuit requirements. We keep the CS policy, but consider finite sources as well as Poisson sources, and the multi-rate generalization. Furthermore, we allow multiple classes with different multi-rate requirements. We considered this example for Poisson arrivals in Section 9 of [6]. Now we consider the effect of finite sources.

The r traffic classes are divided among the six routes as follows: Define nonnegative integers r_i for $i = 0, 1, \ldots, 6$ such that $1 \equiv r_0 < r_1 < r_2 \cdots < r_6 \equiv r$. Class j goes over route l if $r_{l-1} + 1 \le j \le r_l$ for $1 \le l \le 6$. For this generalized Kelly example, the generating function in the finite-source case is given by

$$\begin{aligned} G(z) = {} & \frac{1}{\prod_{i=1}^{6}(1 - z_i)} \prod_{j=1}^{r_1}(1 - p_j + p_j z_1^{a_{1j}})^{N_j} \\ & \times \prod_{j=r_1+1}^{r_2}(1 - p_j + p_j z_2^{a_{2j}})^{N_j} \\ & \times \prod_{j=r_2+1}^{r_3}(1 - p_j + p_j z_1^{a_{1j}} z_2^{a_{2j}})^{N_j} \\ & \times \prod_{j=r_3+1}^{r_4}(1 - p_j + p_j z_3^{a_{3j}} z_5^{a_{5j}})^{N_j} \\ & \times \prod_{j=r_4+1}^{r_5}(1 - p_j + p_j z_4^{a_{4j}} z_5^{a_{5j}})^{N_j} \\ & \times \prod_{j=r_5+1}^{r}(1 - p_j + p_j z_1^{a_{1j}} z_3^{a_{3j}} z_5^{a_{5j}})^{N_j}. \end{aligned} \tag{12.1}$$

As in [6], the dimension can be reduced from 5 to 3 by designating z_1 and z_5 as initial variables to invert. For any given $(z_1,\ z_5)$, the generating function $G(\mathbf{z})$ can be written as the product of three factors, each involving only one of the remaining variables; i.e., the optimal order of inversion is z_1, z_5, z_2, z_3 and z_4. Thus, the inversion dimension is reduced from 5 to 3. For the optimal inversion order, we use the roundoff-error-control l_j parameter vectors $(1,2,3,3,3)$ and $(1,3,3,3,3)$ in the inversion. (See Section 3.2 of [6].)

The specific example we consider has capacities $K_i = 15$ for each i, $1 \le i \le 5$. There are 12 classes, with two classes using each of the 6 routes. We use this example to show how the finite sources approach Poisson sources as N_j gets large and p_j decreases with $N_j p_j = \rho_j$ (constant) for each j. We consider Poisson arrivals and two cases of finite sources, one with $p_j = 10^-3$ for all j and the other with $p_j = 10^{-1}$ for all j. We choose N_j so that the given Poisson model is the natural Poisson approximation for the finite-source models. The specific offered loads are given in Table 1. We let the requirements be either 1 or 2 for each request, with each request having the same requirements on each link. The specific requirements are also given in Table 1.

The blocking probabilities for each class in these three cases are given in Table 1. As anticipated, the finite-source blocking probabilities are quite close to the Poisson blocking probabilities, especially when $p = 10^{-3}$. The computation of the blocking probabilities in Table 4 took about 1 minute. For this specific example, we could allow some of the resource capacities to be much larger through the use of truncation. However, the inversion algorithm will encounter difficulties for even larger networks without special structure.

13 An Example Illustrating Elimination of Very Lightly Loaded Resources

The Kelly example above illustrates dimension reduction by conditional decomposition of the generating function, i.e., by inverting the variables in a good order. Now we illustrate dimension reduction by initially eliminating resources that are so lightly loaded that they impose essentially no constraint, as proposed in Section 7. In some cases, it is obvious that resources impose essentially no constraint, but in other cases it is not. Here it is not so obvious. In general, it seems desirable to systematically exploit the normal approximations in Section 7 for this purpose, as we do here. Afterwards, we would look for further dimension reduction by conditional decomposition.

Our example starts with 8 resources serving 8 types of job classes, but it turns out that only 3 of the 8 resources actually need to be considered. Moreover, each type uses at most 1 of these 3 remaining resources, so that all final required inversions are only 1-dimensional. Thus, this is a best-case scenario, but it seems realistic.

The initial 8×8 requirements matrix is given in Table 2 and the remaining model parameters are given in Table 3. The positive entries in each column of the requirements matrix are identical, as required by our assumptions for efficiently treating guaranteed minimum parameters; see Section 3.

Each type has multiplicities, so that there are more than 8 classes. Indeed, there are 490 classes in all. We consider the unbuffered variant with BPP arrivals, where the arrival process is specified by the (average) arrival rate $\overline{\lambda}_j \equiv \alpha_j + \beta_j m_j$ and peakedness $z_j \equiv v_j/m_j$, using (6.1). The associated BPP parameters are given by (6.2). For simplicity, we let the mean job holding time for each class be 1.

Each type has upper limit and guaranteed minimum bounds. Class 1 has a premium grade of service with a guaranteed minimum twice the specified rate. Hence, the mean capacity used for class 1 is (approximately) this guaranteed minimum bound itself and the variance is approximately 0.

Class 4 and 5 have the next highest grades of service. Their GM parameters are set equal to their arrival rates. We thus apply Theorem 7.1 to determine the means and variances of the capacity used by each class. In particular, we apply (7.11) and (7.12).

The remaining classes have sufficiently low GM parameters and sufficiently high UL parameters that they have negligible effect on the capacity used (assuming that these classes send traffic according to the specified parameters). Thus, for these classes, the means and variances are given by $m_j = \overline{\lambda}_j$ and $v_j \equiv m_j z_j$. The last two rows of Table 3 give the means and variances of the capacity used by all the classes of each type (in the resources they require).

The capacities of the resources are given in Table 4. Also given are the means, variances and standard deviations of the capacity needed in each resource. The means and variances are obtained by adding the values in the last two rows of Table 3 for all classes that use the resource.

For each resource, Table 4 also displays the binding parameter, as specified in (7.2). For comparison, we also display the percentage of the total capacity that is expected to be needed. These percentages range from 78%–100%. The two highest percentages, 95% and 100%, correspond to two of the three resources that are not negligible, but the third relevant resource (resource 4) actually ranks sixth in percentage. This can be explained by the fact that it has smaller capacity. A rough estimate for the threshold of mean capacity needed is $K_i - 3\sqrt{K_i}$. The fraction $(K_i - 3\sqrt{K_i})/K_i = 1 - 3/\sqrt{K_i}$ is larger for larger K_i.

When we eliminate the negligible resources and consider only resources 3, 4 and 8, we see that no type needs more than one of these resources. Thus, we can analyze these three resources separately. Moreover, types 5 and 8 do not need any of these resources. Hence, they should experience essentially no blocking.

We thus can compute all the blocking probabilities using three separate one-dimensional inversions, i.e. we consider types 1, 3 and 7 on resource 3, types 2 and 4 on resource 4, and type 6 on resource 8. These remaining problems are not nearly trivial though. For example, resource 3 with capacity $K_3 = 9,500$ serves 350 classes (of 3 types). Nevertheless, our algorithm requires only a few seconds for each of the three separate problems. The blocking probabilities for each class are given in Table 5. (No calculation is performed for types 5 and 8.)

14 An Example with General State-Dependent Service Rates

We now give an example showing that we can treat general service rates; i.e., showing that the algorithm is not restricted to the unbuffered and single-server buffered variants. Earlier recursions developed for the unbuffered variants no longer apply, but our recursion based on convolution developed in Section 11 does apply. We apply both the inversion and convolution algorithms to this example.

Our example has a single resource with three types of classes, each with Poisson arrivals, an upper limit sharing policy and multiple servers. Multiple servers means that the service rate is proportional to the number of jobs up to a threshold and thereafter it is constant, as in the multi-server M/M/s queue. Here are the parameters:

Class type 1 has 2 servers, offered load $\rho_1 = 1.5$, resource-unit requirement $a_{11} = 3$ and an upper limit of 10 jobs (30 resource units).

Class type 2 has 10 servers, offered load $\rho_2 = 10.0$, resource-unit requirement $a_{12} = 2$ and an upper limit of 50 jobs (100 resource units).

Class type 3 has 40 servers, offered load $\rho_3 = 42$, resource-unit requirement $a_{13} = 1$ and an upper limit of 100 jobs (100 resource units).

In Table 6 we show the results for 4 cases with different multiplicities and capacities.

Note that the computational complexity of the inversion algorithm is $O(\overline{r}\sqrt{K})$ and that of the recursive algorithm is $O(rK^2)$ where K is the number of resource units, r is the number of classes, and $\overline{r}$ is the number of types (3 in the above example). For any two consecutive cases, r and K grow by a factor of 10, while $\overline{r}$ remains constant. So the computational complexity of the inversion algorithm grows by a factor of $\sqrt{10} \simeq 3.16$, while that of the recursive algorithm grows by a factor of $10^3 = 1000$. In fact the recursive algorithm can only compute the first two cases in reasonable time, taking about a minute for the second case. For those two cases, the two algorithms agree closely (more than the displayed 8 digits). We did not apply the recursive algorithm in the last 3 cases since it would have taken about 10^3, 10^6 and 10^9 minutes, respectively. By contrast, the inversion algorithm was applied to all cases and even in the challenging last case took less than half a minute.

Accuracy in the last three cases was verified by doing the inversion twice with inversion parameters $l_1 = 1$ and $l_1 = 2$ (see Section 3.2 of [6]) and the agreement was more than the displayed 8 digits. The high accuracy of this example supports the heuristic scaling in Section 8.

Next we compute derivatives of the blocking probabilities with respect to the offered-load parameters using equations (8.2) and (8.4) of [7]. Due to many possible combinations of derivatives, we show them only for the first case of this example.

The accuracy of these derivative calculations was verified in two ways. First, as usual, we compared the results with inversion parameters $l_1 = 1$ and $l_1 = 2$. Second, we computed finite differences of blocking probabilities and observed that they approach the derivative value as the granularity of the finite difference decreases.

References

[1] J. Abate and W. Whitt. The Fourier-Series Method for Inverting Transforms of Probability Distributions. *Queueing Systems*, 10:5–88, 1992.

[2] J. Abate and W. Whitt. Numerical Inversion of Probability Generating Functions. *Opns. Res. Letters*, 12:245–251, 1992.

[3] D. Y. Burman, J. P. Lehoczky, and Y. Lim. Insensitivity of Blocking Probabilities in a Circuit Switching Network. *J. Appl. Prob.*, 21:850–859, 1984.

[4] G. L. Choudhury, Y. Kogan, and S. Susskind. Exact and Asymptotic Solutions for Models of New Telecommunication Services with Blocking. In *Proc. Third Int. Workshop on Queueing Networks with Finite Capacity*, pp. 29/1–29/10. Ilkley, West Yorkshire, England, 1995.

[5] G. L. Choudhury, K. K. Leung, and W. Whitt. Resource-Sharing Models with State-Dependent Arrivals of Batches. In W. J. Stewart, editor, *Computations with Markov Chains*, pp. 255–282. Kluwer, Boston, 1995.

[6] G. L. Choudhury, K. K. Leung, and W. Whitt. An Algorithm to Compute Blocking Probabilities in Multi-Rate Multi-Class Multi-Resource Loss Models. *Adv. Appl. Prob.*, 1995, to appear. (Abbreviated version in *Proc. IEEE Globecom '94*, pp. 1123–1128, 1994).

[7] G. L. Choudhury, K. K. Leung, and W. Whitt. An Inversion Algorithm for Loss Networks with State-Dependent Rates. *Proc. IEEE Infocom '95*, pp. 513–521, 1995.

[8] G. L. Choudhury, K. K. Leung, and W. Whitt. Calculating Normalization Constants of Closed Queueing Networks by Numerically Inverting Their Generating Functions. *J. ACM*, to appear. (Abbreviated version in *Proc. 1994 Conf. on Inf. Sci. and Systems*, Princeton University, H. Kobayashi ed., pp. 7–11, 1994).

[9] G. L. Choudhury, D. M. Lucantoni, and W. Whitt. Multidimensional Transform Inversion With Applications to the Transient M/G/1 Queue. *Ann. Appl. Prob.*, 4:719–740, 1994.

[10] M. C. Chuah. General Pricing Framework for Multiple Service, Multiple Resource Systems. AT&T Bell Laboratories, Holmdel, New Jersey, 1993.

[11] A. E. Conway and E. Pinsky. A Decomposition Method for the Exact Analysis of Circuit-Switched Networks. *Proc. IEEE Infocom '92*, pp. 996–1003, 1992.

[12] A. E. Conway, E. Pinsky, and S. Tripandapani. Efficient Decomposition Methods for the Analysis of Multi-Facility Blocking Models. *J. ACM*, 41:648–675, 1994.

[13] L. E. N. Delbrouck. A Unified Approximate Evalua-

tion of Congestion Functions for Smooth and Peaky Traffic. *IEEE Trans. Commun.*, 29:85–91, 1981.

[14] L. E. N. Delbrouck. On the Steady-State Distribution in a Service Facility with Different Peakedness Factors and Capacity Requirements. *IEEE Trans. Commun.*, COM-31:1209–1211, 1983.

[15] Z. Dziong and J. W. Roberts. Congestion Probabilities in a Circuit-Switching Integrated Services Network. *Perf. Eval.*, 7:267–284, 1987.

[16] F. Kamoun and L. Kleinrock. Analysis of Shared Finite Storage in a Computer Network Node Environment Under General Traffic Conditions. *IEEE Trans. Commun.*, COM-28:992–1003, 1980.

[17] J. S. Kaufman. Blocking in a Shared Resource Environment. *IEEE Trans. Commun.*, COM-29:1474–1481, 1981.

[18] J. S. Kaufman and K. M. Rege. Blocking in a Shared Resource Environment with Batched Poisson Arrival Processes. *Perf. Eval.*, to appear.

[19] F. P. Kelly. *Reversibility and Stochastic Networks.* Wiley, New York, 1979.

[20] F. P. Kelly. Blocking Probabilities in Large Circuit-Switched Networks. *Adv. Appl. Prob.*, 18:473–505, 1986.

[21] F. P. Kelly. Loss Networks. *Ann. Appl. Prob.*, 1:319–378, 1991.

[22] Y. Kogan and M. Shenfild. Asymptotic Solutions of Generalized Multiclass Engset Model. In J. Labetoulle and J. W. Roberts, editors, *The Fundamental Role of Teletraffic in the Evolution of Telecommunication Networks, Proceedings of the* 14^{th} *Int. Teletraffic Congress*, pp. 1239–1249. Elseiver, Amsterdam, 1994.

[23] J. P. Labourdette and G. W. Hart. Blocking Probabilities in Multitraffic Loss Systems: Insensitivity, Asymptotic Behavior, and Approximations. *IEEE Trans. Commun.*, 40:1355–1366, 1992.

[24] S. S. Lam and Y. L. Lien. A Tree Convolution Algorithm for the Solution of Queueing Networks. *Commun. ACM*, 26:203–215, 1983.

[25] B. Melamed and W. Whitt. On Arrivals That See Time Averages. *Opns. Res.*, 38:156–172, 1990.

[26] D. Mitra and J. A. Morrison. Erlang Capacity and Uniform Approximations for Shared Unbuffered Resources. In J. Labetoulle and J. W. Roberts, editors, *The Fundamental Role of Teletraffic in the Evolution of Telecommunication Networks, Proceedings of the* 14^{th} *Int. Teletraffic Congress*, pp. 875–886. Elsevier, Amsterdam, 1994.

[27] J. A. Morrison. Blocking Probabilities for Multiple Class Batched Poisson Arrivals to a Shared Resource. *Perf. Eval.*, to appear.

[28] E. Pinsky and A. E. Conway. Computational Algorithms for Blocking Probabilities in Circuit-Switched Networks. *Ann. Opns. Res.*, 35:31–41, 1992.

[29] M. Reiser and H. Kobayashi. Queueing Networks with Multiple Closed Chains: Theory and Computational Algorithms. *IBM J. Res. Develop.*, 19:283–294, 1975.

[30] J. W. Roberts. A Service System with Heterogeneous User Requirements. In G. Pujolle, editor, *Perf. of Data Commun. Systems and their Applications*, pp. 423–431. North-Holland Publishing, 1981.

[31] D. Tsang and K. W. Ross. Algorithms to Determine Exact Blocking Probabilities for Multirate Tree Networks. *IEEE Trans. Commun.*, 38:1266–1271, 1990.

[32] H. A. B. van de Vlag and G. A. Awater. Exact Computation of Time and Call Blocking Probabilities in Multi-Traffic Circuit-Switched Networks. *Proc. IEEE Infocom '94*, pp. 56–65, 1994.

[33] E. A. van Doorn and J. F. M. Panken. Blocking Probabilities in a Loss System with Arrivals in Geometrically Distributed Batches and Heterogeneous Service Requirements. *IEEE/ACM Trans. Networking*, 1:664–667, 1993.

[34] A. M. Viterbi and A. J. Viterbi. Erlang Capacity of a Power Controlled CDMA System. *IEEE J. Sel. Areas Commun.*, 11:892–900, 1993.

[35] W. Whitt. Heavy-Traffic Approximations for Service Systems with Blocking. *AT&T Bell Lab. Tech. J.*, 63:689–708, 1984.

[36] W. Whitt. Tail Probabilities with Statistical Multiplexing and Effective Bandwidths in Multi-Class Queues. *Telecommunication Systems*, 2:71–107, 1993.

[37] R. I. Wilkinson. Theories for Toll Traffic Engineering in the U.S.A. *Bell System Tech. J.*, 35:421–507, 1956.

Gagan L. Choudhury received the B.Tech. degree in Radio Physics and Electronics from the University of Calcutta, India in 1979 and the MS and Ph.D. degrees in Electrical Engineering from the State University of New York (SUNY) at Stony Brook in 1981 and 1982, respectively. Currently he is a technical manager at the AT&T Bell Laboratories, Holmdel, New Jersey, USA. His group is responsible for the performance assessment of various AT&T products and services based on modeling and simulation. His main research interest is in the development of multidimensional numerical transform inversion algorithms and their application to the performance analysis of telecommunication and computer systems.

Kin K. Leung (S'78–M'86–SM'93) received the B.S. degree with first class honors in electronics from the Chinese University of Hong Kong, Hong Kong, in 1980, and the M.S. and Ph.D. degrees in computer science from the University of California, Los Angeles, in 1982 and 1985, respectively.

He attended UCLA under an exchange program between the Chinese University of Hong Kong and the University of California. At UCLA, he served as a teaching assistant from 1981 to 1983 and a postgraduate research engineer from 1983 to 1985. Since 1986, he has been working in the Teletraffic Theory and System Performance Department at AT&T Bell Laboratories, Holmdel, NJ. His research interests include stochastic modeling, distributed processing and databases, and wireless and computer communication networks.

Ward Whitt received the A.B. degree in mathematics from Dartmouth College in 1964 and the Ph.D. degree in operations research from Cornell University in 1969.

He taught in the Department of Operations Research at Stanford University in 1968–1969, and in the Department of Administrative Sciences at Yale University from 1969 to 1977. Since 1977, he has been employed by AT&T Bell Laboratories. He currently works in the Network Services Research Laboratory, Murray Hill, NJ. His research interests include queueing theory, stochastic processes, stochastic models in telecommunications and numerical inversion of transforms.

Dr. Whitt is a member of the Operations Research Society of America and the Institute of Mathematical Statistics.

class parameters				blocking probabilities for each class		
j	ρ_j	route	rqmts.	Poisson arrivals	finite source $p_j = 10^{-3}$	finite source $p_j = 10^{-1}$
1	2	1	1	0.069930	0.069934	0.070243
2	1	1	2	0.155912	0.155892	0.153620
3	2	2	1	0.011306	0.011285	0.009104
4	1	2	2	0.030032	0.029961	0.022803
5	2	1,2	1	0.079276	0.079265	0.077959
6	1	1,2	2	0.176021	0.175969	0.170178
7	2	3,5	1	0.071961	0.071961	0.071908
8	1	3,5	2	0.159575	0.159550	0.156668
9	2	4,5	1	0.071961	0.071961	0.071908
10	1	4,5	2	0.159575	0.159550	0.156668
11	2	1,3,5	1	0.134236	0.134246	0.135156
12	1	1,3,5	2	0.280670	0.280668	0.279973

Table 1: Blocking probabilities in the Kelly example with the CS policy.

resource	job type							
	1	2	3	4	5	6	7	8
1	20	10	10	5	5	2	1	1
2	20	10	0	0	5	2	1	0
3	20	0	10	0	0	0	1	0
4	0	10	0	5	0	0	0	0
5	0	10	0	0	5	2	0	1
6	20	0	10	5	0	0	1	1
7	0	0	0	0	5	0	0	1
8	0	0	0	0	0	2	0	0

Table 2: The initial requirements matrix.

parameters for each type	job type 1	2	3	4	5	6	7	8
multiplicity	10	10	40	10	10	100	300	10
arrival rate	10.0	20.0	5.0	10.0	100.0	10.0	10.0	100.0
peakedness	1.0	2.0	1.0	0.8	1.0	1.0	3.0	0.7
mean holding time	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
upper limit	30	40	15	30	150	30	30	180
guaranteed minimum	20	10	1	10	100	0	5	50
mean capacity used for each class	400.	200.	50.	55.7	520.	20.	10.	100.
variance of capacity used for each class	0.	4000.	500.	68.1	851.9	40.	30.	70.
mean capacity used for each type	4000	2000	2000	557	5200	2000	3000	1000
variance of capacity used for each type	0	40,000	20,000	681	8519	4000	9000	700

Table 3: Parameters for the job types.

	resource							
	1	2	3	4	5	6	7	8
capacity K_i	23,000	18,500	9,500	3,000	12,500	13,500	7,000	2,000
mean capacity needed M_i	19,757	16,200	9,000	2,557	10,200	10,557	6,200	2,000
variance of capacity needed V_i	82,900	61,519	29,000	40,681	53,219	30,381	9,219	4,000
standard deviation of capacity needed $\sqrt{V_i}$	288.	248.0	170.	202.	231.	174.	96.	63.
binding parameter γ_i	11.3	9.3	2.9	2.2	10.0	16.9	8.3	0.0
percentage of capacity expected to be needed	86%	88%	95%	85%	82%	78%	89%	100%

Table 4: The capacity of the resources and an initial analysis of the demand.

type	blocking probability	type	blocking probability
1	2.489e-6	5	0.0
2	3.615e-3	6	2.496e-2
3	3.591e-4	7	1.827e-5
4	4.701e-4	8	0.0

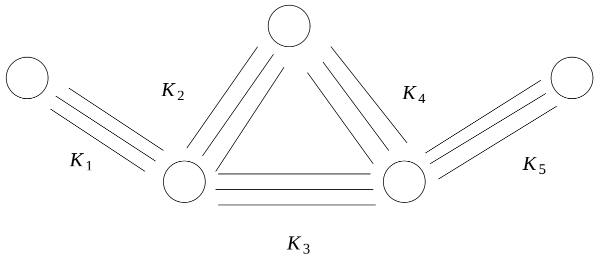
Table 5: Blocking probabilities for the eight customer types. The algorithm has been applied to the three relevant resources separately.

number of classes	multiplicity of each type	number of resource units	blocking probabilities		
			type 1	type 2	type 3
3	1	100	0.17755446	0.11995594	0.06159283
30	10	1000	0.15486823	0.10372589	0.06354832
300	100	10,000	0.15098952	0.10117050	0.06341047
3000	1000	100,000	0.15056962	0.10089765	0.06339184
30,000	10,000	1,000,000	0.15052725	0.10087015	0.06338993

Table 6: Blocking probabilities for the multi-server example.

$\frac{dB_i}{d\rho_j}$	$i = 1$	$i = 2$	$i = 3$
$j = 1$	0.027234283	0.010284699	0.005366907
$j = 2$	0.0212418781	0.015484701	0.008190657
$j = 3$	0.036043304	0.026384510	0.014137514

Table 7: Derivatives for the first case of the multi-server example.



K_1
K_2
K_3
K_4
K_5