AN INVERSION ALGORITHM FOR LOSS NETWORKS WITH STATE-DEPENDENT RATES

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

[1] AT&T Bell Laboratories, Room 1L-238, Holmdel, NJ 07733-3030
[2] AT&T Bell Laboratories, Room 1L-215, Holmdel, NJ 07733-3030
[3] AT&T Bell Laboratories, Room 2C-178, Murray Hill, NJ 07974-0636

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. Major features of the algorithm are: dimension reduction by conditional decomposition based on special structure, an effective scaling algorithm to control errors in the inversion, the efficient treatments of multiple classes with identical parameters and the truncation of large sums.

1. Introduction

In this paper we describe a new algorithm to compute blocking probabilities for each class and other steady-state performance measures in a family of product-form loss networks (or resource-sharing models). 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 [11], 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.) Here, however, we consider a more general model with *state-dependent arrival and service rates.* 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 [5,6] 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. Considerable research has been focused on the unbuffered variant, but an effective algorithm for the buffered variant has been developed previously only in the special case of single constant-rate server per class and with a number of other restrictions [9]. So a significant contribution of this paper is the effective treatment of buffered models.

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 [10]). 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.

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,

0743-166X/95 $04.00 © 1995 IEEE

extending [5,6,7], by using state-dependent arrival rates to represent overflow traffic associated with alternative routing. Networks of moderate size or with special structure allowing dimension reduction (see [1,2,3]) can be treated by our method exactly. Large networks without special structure can be analyzed approximately by extending the reduced-load fixed-point approximations [11].

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 can show 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. This is demonstrated in (5.3) for the mean, but it is true for other steady-state quantities as well.

Our algorithm here extends our previous algorithm for loss networks without state-dependent rates in [1,2]. It is based on deriving a convenient expression for the generating function of the normalization constant and then numerically inverting the generating function. 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 inverting the variables in a good order. For example, this 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 [12] for closed queueing networks and by Conway, Pinsky and Tripandapani [4] for special cases of models 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.

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 [17] and emerging high-speed communication networks employing the asynchronous transfer mode (ATM) technology [13]. 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.

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 [5,6] and Dziong and Roberts [7] 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 7). 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. For a single resource, the UL policy is equivalent to the tree networks considered by Tsang and Ross [15] (for the case of Poisson arrivals).

1.2 The Buffered Variant

The buffered variant with a single constant-rate server per class was considered by Kamoun and Kleinrock [9] 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 the non-complete-sharing policies. In Section 4 we discuss how to compute blocking probabilities. In Section 5 we discuss modeling with BPP arrival processes (the connection to peakedness). In Section 6 we describe the new scaling algorithm. Section 7 discusses the computational complexity of our algorithm. Section 8 shows how to study the sensitivity of blocking probabilities to traffic parameters. Finally in Sections 9 and 10 we present illustrative numerical examples. More details appear in an expanded version of this paper [3].

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,* $1 \le i \le p$, 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,

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

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.

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

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

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 $\bar{K}_i = \sum_{j=1}^{r} a_{ij} n_j$. Then

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

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

$n_j \geq N_j$ which also implies $f_j(n_j) = 0$ for $n_j \geq N_j$. This gives

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

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

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

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

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

which is the same as (2.12) of [2].

If, instead, $\lambda_j(k) = \alpha_j + \beta_j k$ where $\beta_j \neq 0$, as in the binomial and Pascal (negative binomial) cases of the BPP model of [6,7], then

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

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

$$G_j(\mathbf{z}) = \sum_{n_j=0}^{\infty} \binom{r_j+n_j-1}{r_j-1} \left[\frac{\beta_j}{\mu_j} \prod_{i=1}^{p} z_i^{a_{ij}}\right]^{n_j}$$

$$= \left[1 - \frac{\beta_j}{\mu_j} \prod_{i=1}^{p} z_i^{a_{ij}}\right]^{-r_j} . \qquad (2.13)$$

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 $\beta_j \geq \mu_j$ 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

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

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

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

Combining (2.8) with (2.15), we get

$$G_j(\mathbf{z}) = \sum_{n_j=0}^{s_j-1} \frac{\left[\rho_j \prod_{i=1}^{p} z_i^{a_{ij}}\right]^{n_j}}{n_j!}$$

$$+ \frac{\left[\rho_j \prod_{i=1}^{p} z_i^{a_{ij}}\right]^{s_j}}{s_j!} (1-\rho_j \prod_{i=1}^{p} z_i^{a_{ij}/s_j})^{-1} . \qquad (2.16)$$

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

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

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 and the $r_j = 1$ and $\beta_j/\mu_j = 1$.

3. Other Sharing Policies

We can introduce other sharing policies by imposing additional constraints on the set of feasible states. As noted in [2], 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 [2], 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. (Clearly the individual UL and GM policies are special cases.) As in [2], 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 *units* guaranteed for class j jobs and let $\mathbf{N} = (N_1, N_2, \ldots, N_r)$.

Let L_{ij} be the upper limit on the number of class j jobs allowed to simultaneously use resource i. 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 and let $\mathbf{M} = (M_1, M_2, \ldots, M_r)$.

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

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

and

$$S_{GM}(\mathbf{K},\mathbf{N}) =$$

$$\{\mathbf{n}\in\mathbf{Z}_+^r : \sum_{j=1}^{r}(a_{ij}n_j \vee \delta_{ij}N_j) \le K_i,\ 1 \le i \le p\} \qquad (3.3)$$

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 $\sum_{j=1}^{r} N_j \le K_i$. 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

$$G(\mathbf{z},\mathbf{y},\mathbf{x}) =$$

$$\sum_{K_1=0}^{\infty}\cdots\sum_{N_r=0}^{\infty} g(\mathbf{K},\mathbf{M},\mathbf{N})\, z_1^{K_1}\cdots z_p^{K_p}\, y_1^{M_1}\cdots y_r^{M_r}\, x_1^{N_1}\cdots x_r^{N_r} \qquad (3.4)$$

In [3] we show that it is possible to get an explicit expression for $G(\mathbf{z},\mathbf{y},\mathbf{x})$. However, it is computationally difficult to invert it since it has $p+2r$ dimensions. In [3] we show that, by virtue of the special structure, it is possible to explicitly invert with respect to all the x and y variables and arrive at the following remarkably simple final expression

$$G(\mathbf{z},\mathbf{M},\mathbf{N}) = \prod_{i=1}^{p}(1-z_i)^{-1}\prod_{j=1}^{r} G_j(\mathbf{z},M_j,N_j)\ , \qquad (3.5)$$

where

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

$$+ \sum_{n_j=\lfloor N_j/b_j\rfloor+1}^{M_j} f_j(n_j)\left(\prod_{i=1}^{p} z_i^{\delta_{ij}}\right)^{n_j b_j}\ . \qquad (3.6)$$

Note that (3.5) and (3.6) represent only p-dimensional transforms, so that computation of the combined UL/GM policy is almost as fast as for the standard CS policy. The only difference is that with the CS policy often we have closed-form expression for $G_j(\mathbf{z})$, but with combined UL/GM policy that is less likely to happen. However, in the buffered variant with single server per class and Poisson arrivals (i.e., the model in [9]) the sum in (3.6) does have a closed form (see [3]) and hence in that case the computation with combined UL/GM policy is just the same as with the CS policy. This is remarkable since in [9] the computational effort required for the UL/GM policy is much greater than for the CS policy, and there are additional restrictions.

All the unbuffered and buffered variants considered in Section 2 are easily obtained by inserting in the corresponding expressions for $f_j(n_j)$ in (3.6). Note that the overall generating function, (2.7) or (3.5), 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.

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.

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.

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 as is clear from the following theorem, which is proved in [3].

Theorem 4.1. *The class-j blocking probability B_j coincides with the time-blocking quantity B_j^i in (4.1) for the modified model in which the class-j arrival-rate function is changed from $\lambda_j(m)$ to $\bar{\lambda}_j(m) \equiv \lambda_j(m+1)$.*

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

$$\bar{\lambda}_j(m) = \lambda_j(m+1) = (\alpha_j + \beta_j) + \beta_j m\ , \qquad (4.2)$$

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 [7], p. 273. Van de Vlag and Awater [16] have recently developed an efficient procedure for computing call blocking probabilities for many classes. However, both of the above only considered the CS policy.

5. Using BPP Arrivals to Model Non-Poisson Traffic

The previous section assumed that the sources really are of type BPP, as arises in finite-source models or in models with controlled arrival rates. However, another important use of the BPP model is to represent non-Poisson traffic, which occurs in overflow traffic associated with alternative routing [5,6,7]. Non-Poisson traffic can be characterized approximately via a *peakedness* parameter [8]. 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

$$M_j = \alpha_j/(\mu_j - \beta_j) \text{ and } V_j = \mu_j\alpha_j/(\mu_j - \beta_j)^2 . \quad (5.1)$$

From (5.1), we see that the two BPP parameters for each class can be expressed as

$$\alpha_j = M_j\mu_j/z_j \text{ and } \beta_j = \mu_j(z_j - 1)/z_j \quad (5.2)$$

where $z_j \equiv V_j/M_j$ is the peakedness.

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 [5], 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 m_j, the mean number of class-j jobs in service in the actual system with capacity constraints. Note that Delbrouck [5] computed m_j in a much simpler system with single rate, single resource and CS policy. We show for the first time here that simple expressions for m_j exist even in our much more general model. Specifically, with combined UL/GM policy,

$$m_j = \frac{\alpha_j \bar{g}(\mathbf{K}-\mathbf{A}\mathbf{e}_j, \mathbf{M}-\mathbf{e}_j, \mathbf{N}-\mathbf{A}\mathbf{e}_j)}{\mu_j g(\mathbf{K},\mathbf{M},\mathbf{N})} , \quad (5.3)$$

where $\mathbf{e}_j$ is a vector with a 1 in the j^{th} place and 0's elsewhere, and the symbol $\bar{g}$ in the numerator of (5.3) 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

$$B_j = 1 - \frac{m_j}{M_j} \quad (5.4)$$

$$= 1 - \left[1 - \frac{\beta_j}{\mu_j}\right] \frac{\bar{g}(\mathbf{K}-\mathbf{A}\mathbf{e}_j, \mathbf{M}-\mathbf{e}_j, \mathbf{N}-\mathbf{A}\mathbf{e}_j)}{g(\mathbf{K},\mathbf{M},\mathbf{N})} .$$

6. A New Scaling Algorithm

Most of the algorithm is as in [1,2], 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 Fourier-series method. We scale the generating function in each step by defining a *scaled generating function* as

$$\bar{G}_j(z_j) = \alpha_{0j} G_j(\alpha_j z_j) , \quad (6.1)$$

where α_{0j} and α_j are positive real numbers. We invert this scaled generating function after choosing α_{0j} and α_j so that the *aliasing error* is suitably controlled.

We propose a new heuristic scaling algorithm which applies to generating functions of any form and hence is applicable to the general state-dependent arrival and service rates. Here we describe the heuristic for only one dimension. For further discussion, see [1,2,3]. Let,

$$G(z) = (1-z)^{-1} \prod_{j=1}^{r} G_j(z) \quad (6.2)$$

with

$$G_j(z) = \sum_{n_j=0}^{\infty} f_j(n_j)(z^{a_j})^{n_j} . \quad (6.3)$$

Then α is the largest number in the interval $(0,1]$ such that

$$\sum_{j=1}^{r} \frac{zG_j'(z)}{G_j(z)}\Big|_{z=\alpha} \le K , \quad (6.4)$$

where $G_j'(z) = \frac{d}{dz}G_j(z)$. Then

$$\alpha_0^{-1} = \prod_{j=1}^{r} G_j(\alpha) . \quad (6.5)$$

This scaling agrees with the previous scalings in the special cases treated in detail in [1,2].

7. Computational Complexity

For simplicity, assume that the capacity of each resource is K and the number of terms in sums of the form (2.8) or (3.6) is M. Then it can be shown [3] that the computational complexity of our algorithm is

$$C = O(M\bar{r}\bar{K}^{\bar{p}}) , \quad (7.1)$$

where

M = 1 if the sum has closed form expression (true for most previous algorithms considered in the literature as well as many new ones we introduce).

$\bar{K} \le K$ and $\bar{K} = O(\sqrt{K})$ for large K

$\bar{p} \le p$ and $\bar{p} << p$ with special structure

$\bar{r} \le r$ and $\bar{r} << r$ for large r and large multiplicities.

In the special case of the CS policy and unbuffered variant (M = 1), Dziong and Roberts [7] developed an algorithm (generalization of Delbrouck [6]) which has complexity

$$C = O(rK^{2p}) . \quad (7.2)$$

Van de Vlag and Awater [16] improved on that, obtaining complexity

$$C = O(rK^{p}) . \quad (7.3)$$

Comparing (7.1) to (7.2) and (7.3), we see that our computational complexity is lower. However, our major contribution is being able to treat many new and important models.

8. Sensitivity with Respect to Traffic Parameters

In addition to the blocking probabilities, we may also be interested in the sensitivity of these probabilities to small perturbations in one or more of the traffic parameters. These sensitivities can be determined by calculating derivatives. Derivatives can be readily calculated due to the particular structure of our generating functions. We illustrate using the CS policy, but the same procedure works for other policies as well.

Taking logarithms in (4.1) (and disregarding M and N since we have the CS policy), we get

$$\ln(1-B_j^t) = \ln g(\mathbf{K}-\mathbf{a}_j) - \ln g(\mathbf{K}) \ . \qquad (8.1)$$

Differentiating (8.1) yields

$$\frac{dB_j^t}{d\rho_i} = (1-B_j^t)\left[\frac{\dfrac{dg(\mathbf{K})}{d\rho_i}}{g(\mathbf{K})} - \frac{\dfrac{dg(\mathbf{K}-\mathbf{a}_j)}{d\rho_i}}{g(\mathbf{K}-\mathbf{a}_j)}\right], \qquad (8.2)$$

where ρ_i is a traffic parameter for job class i.

From (8.2), we see that it suffices to be able to compute $dg(\mathbf{K})/d\rho_i$. Let I represent the inversion operator. From (2.7) and (2.8),

$$g(\mathbf{K}) = I\Big(\prod_{l=1}^{p}(1-z_l)^{-1}\prod_{j=1}^{r}\sum_{n_j=0}^{\infty}f_j(n_j)\Big(\prod_{l=1}^{p}z_l^{a_{lj}}\Big)^{n_j}\Big) \ . \qquad (8.3)$$

Differentiating both sides with respect to ρ_i, we get

$$\frac{dg(\mathbf{K})}{d\rho_i} = I\Big[\prod_{l=1}^{p}(1-z_l)^{-1}\Big[\prod_{\substack{j=1\\ j\neq i}}^{\infty}\sum_{n_j=0}^{\infty}f_j(n_j)\Big(\prod_{l=1}^{p}z_l^{a_{lj}}\Big)^{n_j}\Big]$$

$$\times\Big[\sum_{n_i=0}^{\infty}f_i'(n_i)\Big(\prod_{l=1}^{p}z_l^{a_{lj}}\Big)^{n_j}\Big]\Big] \ , \qquad (8.4)$$

where $f_i'(n_i) = df_i(n_i)/d\rho_i$, since only the i^{th} term in the numerator of $g(\mathbf{K})$ depends on ρ_i. Hence, the computation of the derivative requires inverting another generating function of nearly the same form.

9. Examples with Complete Sharing

We now give examples illustrating the numerical inversion algorithm. All our examples here have a single resource. Multi-resource examples are discussed in [1,2,3]. All computations were done on a SUN SPARC-2 workstation. Computation times were always a few seconds or less.

Our first numerical example is the classical resource-sharing model with the CS sharing policy and a finite-source input. The generating function is given in (2.7) and (2.14) with $p = 1$. This example is relatively elementary since the generating function is one-dimensional. In [2] we considered this example with Poisson arrivals.

The specific model we consider has 2 classes with requirements $a_{1,1} = 1$ and $a_{1,2} = 12$. The capacity K ranges from 20 (relatively small) to 50,000 (relatively large). The specific parameters we use are $p_1 = 0.3$, $p_2 = 0.1$, $N_1 = 0.5K$ and $N_2 = K$. (See (2.13).) We give numerical results for the blocking probabilities of each class in Table 1.

As a basis for comparison, we also implemented the recursive algorithm of Delbrouck [6] for finite sources, and the uniform asymptotic approximation (UAA) of Mitra and Morrison [13]. In the cases with $K \leq 500$, the inversion and recursion algorithms agreed well beyond the accuracy given (eight significant digits). For $K \geq 5000$, the recursions either had numerical underflows or overflows, or took too long to run. (The recursion for finite sources takes much longer than the recursion for Poisson sources, since the computational complexity is $O(K^2)$ instead of $O(K)$.) For $K \geq 5000$, the inversion results agreed closely with UAA, and, as expected, the agreement improves as K increases. In each case we also checked the accuracy of the inversion algorithm directly by running it twice, with roundoff control parameters $l_1 = 1$ and $l_1 = 2$.

In all cases considered here the inversion algorithm took less than half a second. For the larger values of K, a critical factor in achieving this speed is truncation [2].

For the example in Table 1, the exact blocking probability for class 2 (which requires 12 units per request) is monotonically decreasing in K, but this is not so for class 1. Note that UAA does not capture this non-monotonicity for class 1.

For the example in Table 1, it would suffice to use only the recursions and UAA, using UAA after the recursions take too long. However, it is not difficult to construct examples that are sufficiently large so that the recursion breaks down and yet UAA is not sufficiently accurate (because size alone does not imply that the model is in the proper region for the UAA asymptotics). One way is to make the numbers of sources very unequal in the finite-source case. To illustrate, consider the example in Table 1 with the source populations $N_1 = 5$ and N_2 very large. Numerical results for 4 cases are given in Table 2. When the capacity is 1193 or 11993, there is always room for all 5 class 1 requests, so that there is no blocking for class 1, but when K is a multiple of 12 such as 1200 or 12,000, there is substantial blocking for class 1 when N_2 is large. The lumpiness that UAA does not capture is obviously not gone even when $K = 12{,}000$.

For the smaller capacities 1193 and 1200, we were able to use the Delbrouck recursion to validate our answers (taking several minutes of CPU time), but since the computational complexity of the recursion is $O(K^2)$, we could not do this for the larger capacities. In the latter case we used the built-in accuracy check of the inversion, comparing values with inversion parameters $l_1 = 1$ and $l_1 = 2$.

10. Different Sharing Policies and State-Dependent Rates

Here we give an example containing all the variations considered in the paper, but in the context of a single resource. We combine the unbuffered and buffered variant in this example primarily to illustrate the capability of the algorithm. However, it is worth noting that the combination could arise;

e.g., we might have the standard unbuffered variant except that one class might require access to a database, which operates like a single-server queue and is the critical resource.

In this example there are 4 types of classes each with multiplicity N, so that there are a total of $4N$ classes. We let N range from 1 to 1000. (See first column of Table 3.) Here are the characteristics of the four types:

Type 1 is the unbuffered variant with finite-source (binomial) input having parameters $r_1 = -40$ and $\beta_1/\mu_1 = -0.2$. This corresponds to 40 sources and peakedness of 0.843 (see Section 5). Each job requires only one resource unit; i.e., $a_{11} = 1$. We use a pure UL policy with an upper limit of 200.

Type 2 is the unbuffered variant with Poisson arrivals having parameters $\rho_2 = 10.0$ and $a_{12} = 2$. We use a pure UL sharing policy with an upper limit of 15 job requests (30 resource units).

Type 3 is the unbuffered variant with Pascal arrival process having parameters $r_3 = 25.0$ and $\beta_3/\mu_3 = 0.5$. This corresponds to a peakedness of 2.0. Each request requires 3 resource units; i.e., $a_{13} = 3$. We use both an upper limit and a guaranteed minimum. The UL and GM parameters are 200 and 6 requests (600 and 18 resource units).

Type 4 is the buffered variant with Poisson arrivals having parameters $\rho_4 = 0.9$ and $a_{14} = 4$. It too has both an upper limit and a guaranteed minimum. The UL and GM parameters are 8 and 2 requests (32 and 8 resource units).

The numerical results are shown in Table 3. We start with a small example with 4 classes and 100 resource units and proceed towards a large example with 4000 classes and 100,000 resource units. The last example runs in several seconds. We take advantage of truncation and multiplicity in reducing the computation time. We check accuracy by comparing values with inversion parameters $l_1 = 1$ and $l_1 = 2$. The agreement is more than the displayed number of figures.

References

[1] G. L. Choudhury, K. K. Leung and W. Whitt, "An Algorithm for Product-Form Loss Networks Based on Numerical Inversion of Generating Functions," *Proc. IEEE Globecom '94,* 1123-1128.

[2] 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.

[3] G. L. Choudhury, K. K. Leung and W. Whitt, "An Inversion Algorithm to Compute Blocking Probabilities in Loss Networks with State-Dependent Rates," 1994, submitted.

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

[5] L. E. N. Delbrouck, "A Unified Approximate Evaluation of Congestion Functions for Smooth and Peaky Traffic," *IEEE Trans. Commun.* vol. 29, pp. 85-91, 1981.

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

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

[8] A. E. Eckberg, Jr., "Generalized Peakedness of Teletraffic Processes," *Proc. 10th Int. Teletraffic Congress,* Montreal, Canada, paper 4.4b.3.

[9] F. Kamoun and L. Kleinrock, "Analysis of Shared Finite Storage in a Computer Network Node Environment Under General Traffic conditions. *IEEE Trans. Commun.* vol. 28, pp. 992-1003, 1980.

[10] J. S. Kaufman, "Blocking in a Shared Resource Environment," *IEEE Trans. Commun.* vol. 29, pp. 1474-1481, 1981.

[11] F. P. Kelly, "Loss Networks," *Ann. Appl. Prob.,* vol. 1, pp. 319-378, 1991.

[12] S. S. Lam and Y. L. Lien, "A Tree Convolution Algorithm for the Solution of Queueing Networks," *Commun. ACM,* vol. 26, pp. 203-215, 1983.

[13] D. Mitra and J. A. Morrison, "Erlang Capacity and Uniform Approximations for Shared Unbuffered Resources," *The Fundamental Role of Teletraffic in the Evolution of Telecommunication Networks, Proceedings of the 14th Int. Teletraffic Congress,* J. Labetoulle and J. W. Roberts (eds.), Elsevier, Amsterdam, pp. 875-886, 1994.

[14] J. W. Roberts, "A Service System with Heterogeneous User Requirements," *Perf. of Data Commun. Systems and their Applications,* G. Pujolle (Ed.), North-Holland, Amsterdam, pp. 423-431, 1981.

[15] D. Tsang and K. W. Ross, "Algorithms to Determine Exact Blocking Probabilities for Multirate Tree Networks," *IEEE Trans. Commun.,* vol. 38, pp. 1266-1271, 1990.

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

[17] A. M. Viterbi and A. J. Viterbi, "Erlang Capacity of a Power Controlled CDMA System," *IEEE J. Sel. Areas Commun.,* vol. 11, pp. 892-900, 1993.

parameters			blocking probability of class 1			blocking probability of class 2		
K	N_1	N_2	inversion	recursion	UAA	inversion	recursion	UAA
20	10	20	2.8506801e-4	same	6.73e-2	0.6785863	same	0.6670
50	25	50	8.4188062e-3	same	5.23e-2	0.5446764	same	0.5242
500	250	500	3.4698631e-2	same	3.481e-2	0.3532091	same	0.353138
5,000	2,500	5,000	3.1096495e-2	—	3.1104e-2	0.3163444	—	0.3163388
50,000	25,000	50,000	3.0629867e-2	—	3.06306e-2	0.3116297	—	0.3116291

Table 1. Numerical results for the example in Section 7 with two classes, the CS policy and finite sources.

parameters			blocking probability for class 1	
K	N_1	N_2	inversion	UAA
1193	5	2400	0.0000000	0.0762986
1200	5	2400	0.2742085	0.0758303
11993	5	24000	0.0000000	0.0753203
12000	5	24000	0.2721590	0.0752734

Table 2. Numerical results for the example in Section 7 with two classes, the CS policy and very unequal finite sources.

number of classes	multiplicity of each class-type	number of resource units	blocking probabilities			
			type 1	type 2	type 3	type 4
4	1	100	.05595671	.12483684	.17537541	.14332934
40	10	1000	.03710717	.09479823	.10925576	.11686444
400	100	10,000	.03307838	.08842370	.09616393	.11164371
4000	1000	100,000	.03256304	.08760934	.09451351	.11098258

Table 3. Numerical results for the four-type example in Section 8.