AN ALGORITHM TO COMPUTE BLOCKING PROBABILITIES IN MULTI-RATE MULTI-CLASS MULTI-RESOURCE LOSS MODELS

by

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

AT&T Bell Laboratories

November 8, 1994

To appear in *Advances in Applied Probability*

[1] AT&T Bell Laboratories, Holmdel, NJ 07733-3030

[2] AT&T Bell Laboratories, Murray Hill, NJ 07974-0636

Abstract

In this paper we consider a family of product-form loss models, including loss networks (or circuit-switched communication networks) and a class of resource-sharing models. There can be multiple classes of requests for multiple resources. Requests arrive according to independent Poisson processes. The requests can be for multiple units in each resource (the multi-rate case, e.g., several circuits on a trunk). There can be upper-limit and guaranteed-minimum sharing policies as well as the standard complete-sharing policy. If all the requirements of a request cannot be met upon arrival, then the request is blocked and lost. We develop an algorithm for computing the (exact) steady-state blocking probability of each class and other steady state descriptions in these loss models. The algorithm is based on numerically inverting generating functions of the normalization constants. In a previous paper we introduced this approach to product-form models and developed a full algorithm for a class of closed queueing networks. The inversion algorithm promises to be even more useful for loss models than for closed queueing networks because fewer alternative algorithms are available for loss models. Indeed, for many loss models with sharing policies other than traditional complete sharing, our algorithm is the first effective algorithm. Unlike some recursive algorithms, our algorithm has a low storage requirement. To treat the loss models here, we derive the generating functions of the normalization constants and develop a new scaling algorithm especially tailored to the loss models. In general, the computational complexity grows exponentially in the number of resources, but the computation can often be reduced dramatically by exploiting conditional decomposition based on special structure and by appropriately truncating large finite sums. We illustrate our numerical inversion algorithm by applying it to several examples. To validate our algorithm on small models, we also develop a direct algorithm. The direct algorithm itself is of interest, because it tends to be more efficient when the number of resources is large, but the number of request classes is small. Furthermore, it too allows a form of conditional decomposition based on special structure.

Key words: product-form models, loss networks, circuit-switched communication networks, resource-sharing models, coordinate-convex resource-sharing policies, blocking probabilities, generating functions, numerical transform inversion, normalization constants, partition functions

AMS subject classification: Primary 60K25, Secondary 90B15, 90B22, 60K35.

1. Introduction

In this paper we develop an algorithm for calculating the (exact) blocking probabilities (and other steady-state descriptions) in a family of loss models. These models are multi-dimensional generalizations of the classical Erlang loss model. They all have product-form steady-state distributions. The blocking probabilities have simple expressions in terms of normalization constants (or partition functions). To calculate the blocking probabilities and other steady-state descriptions, we calculate the normalization constants by numerically inverting their generating functions (which we derive here).

Our family of models includes two familiar classes of models that have received considerable attention because of their important application to communication networks and computer systems: (1) *loss networks* (or circuit-switched communication networks) as reviewed by Kelly (1991), and (2) *resource-sharing models* as considered by Kaufman (1981), Roberts (1981) and others. However, our models are more general than are usually considered in these two classes, as we explain next.

1.1 Loss Networks

A loss network has multiple nodes connected by *trunks* (or links), each of which contains a number of *circuits,* as depicted in Figure 1. In this figure there are 5 nodes and 5 trunks, with trunk j having K_j circuits. (The nodes actually play no role in the model.) The loss network carries multiple *classes or types of calls,* which are distinguished by the set of trunks (or *route)* they require, by the number of circuits required on each trunk (which need not be the same on all trunks) and by the average call holding time. (Each call holds all its circuits for the duration of the call.)

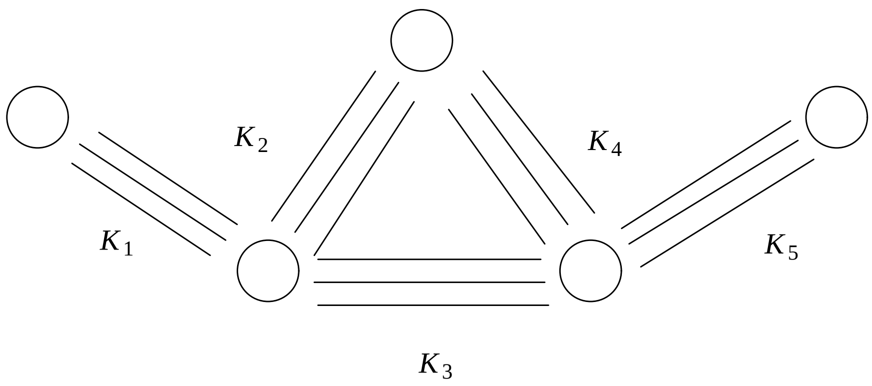

Figure 1. An example of a loss network

For the example in Figure 1, the set of routes might be

$$5 = \{\{1\}, \{2\}, \{1,2\}, \{3,5\}, \{4,5\}, \{1,3,5\}\} .$$

In this example there might be 12 call types, two corresponding to each route (subset) with the route indicating the trunks needed for each call. We also must specify the number of circuits needed by each call type on each trunk. We may have more call types than routes, because different call types with the same route may have different circuit requirements. The steady-state distribution depends on only one more parameter for each call type: the offered load, which is the call arrival rate multiplied by the mean call holding time.

In the basic loss network model, calls of each type arrive according to a Poisson process. Each call is accepted only if all the trunks on its route have enough circuits available to support the call; otherwise, the call is blocked and lost (without generating retrials or otherwise affecting future arrivals). Loss networks have many applications; e.g., a loss network may represent a database, a wireless communication network or a circuit-switched computer network as well as a circuit-switched telephone network. For additional background on loss networks and additional references, see Kelly (1985, 1986, 1991).

We have not mentioned the call holding-time distributions, because the product-form models have an *insensitivity property* implying that the steady-state distribution depends on the call holding time distributions only through their means: e.g., see Lam (1977), Kaufman (1981), Burman, Lehoczky and Lim (1984) and Whitt (1980, 1985).

The example in Figure 1 is taken from the introduction of Kelly (1986). We apply our algorithm to solve this example in Section 9. There are alternative algorithms that could be applied to this example: a direct algorithm which we introduce in Section 6 and recursive algorithms developed by Dziong and Roberts (1987), Pinsky and Conway (1992) and Conway and Pinsky (1992), but all these alternatives encounter numerical difficulties when the model gets large. We try to address this difficulty with our inversion algorithm, as do Conway and Pinsky (1992) with their recursive algorithm. Effective algorithms for computing the exact blocking probabilities in loss networks previously had been developed only for special cases. For a class of two-hop tree networks, Mitra (1987) performed an asymptotic analysis and developed bounds, while Kogan (1989) developed an algorithm for that model by relating it to a closed queueing network. Other algorithms for tree networks have been developed by Tsang and Ross (1990) and Ross and Tsang (1990). We apply our algorithm to these tree networks in Section 10, and show that it may be applied to more general structures as well. Of course, simulation can also be applied. A related approach is Monte Carlo integration; see Ross and Wang (1992).

We generalize the traditional loss network model in the direction of trunk sharing. The traditional loss network model assumes *complete sharing* (CS) of the circuits on a trunk among all competing traffic classes. However, it is often desirable to consider other sharing policies to provide different grades of service (the so-called platinum, gold and vanilla services) or to protect one traffic class from another. We show how to treat general sharing policies based on linear constraints. In particular, each additional linear constraint is equivalent to having another trunk in the model with the CS policy. This is because, with the CS policy, each trunk introduces a linear constraint. Thus, a p-trunk problem with c extra linear constraints is equivalent to a $(p + c)$-trunk problem with the CS policy.

Unfortunately, however, the computational complexity increases dramatically with the number of trunks, because the dimension of the generating function of the normalization constant equals the number of trunks. Thus we focus on two special forms of sharing for which we can show the computational complexity does not grow much. In particular, we consider two other candidate trunk sharing policies: The first is the *upper limit* (UL) policy, which provides upper limits on the numbers of circuits that can be used by each class on each trunk. The second is the *guaranteed minimum* (GM) policy, which reserves a specified number of circuits on each trunk for each traffic class. Both of these sharing policies can be represented by the addition of linear constraints, with one constraint for each class. However, we exploit the special structure to reduce the effective dimension of the transform. We show that the effective dimension with the UL and GM sharing policies is at most one greater than with the CS sharing policy.

In this paper we only consider the two non-CS sharing policies UL and GM, but our results illustrate what can be done more generally. The various sharing policies can be thought of as restrictions of the state space corresponding to the CS policy. From the general theory of reversible Markov processes, it is known that all the models with such restrictions inherit the product-form distribution. The new steady-state distribution is a truncation and renormalization of the steady-state distribution associated with the CS policy; see Section 1.6 of Kelly (1979). (Of course, the restriction must leave an irreducible Markov process.) Moreover, closed-form generating functions of normalization constants can be derived more generally.

Some recent papers on loss networks have focused on extensions which do *not* have product-form steady-state distributions, and thus are beyond the scope of our algorithm. A major goal of these more general models is to represent schemes for providing alternative routes to blocked calls; e.g., see Kelly (1991), Mitra, Gibbens and Huang (1993) and Ash and Huang (1993). Both product-form and non-product-form loss networks have been analyzed primarily by approximate methods, such as *reduced-load or Erlang fixed-point approximations;* see Whitt (1985), Dziong and Roberts (1987) and Chung and Ross (1993) in addition to the references above. Simulation has also been applied, see Greenberg, Nicol and Lubachevsky (1992) and Gaujal, Greenberg and Nicol (1993). These alternative methods are often effective, but there remains a need for exact algorithms such as we develop here. Moreover, our algorithm for product-form models can assist in developing and evaluating approximations for more complicated non-product-form models. New reduced-load approximations for large loss networks can be based on the exact solution for single links or more general subnetworks with Poisson arrivals.

1.2 Resource-Sharing Models

The second class of models we consider are the *resource-sharing models.* These models are closely related to the loss networks, but they have evolved in a somewhat separate literature. The basic resource-sharing model has a *single resource,* such as a set of memory buffers, which is shared by several classes of requests. A distinguishing feature of the resource-sharing model is the use of *resource-sharing policies,* such as the UL and GM policies discussed above; see Kamoun and Kleinrock (1980) and Kaufman (1981).

One version of the resource-sharing model can be regarded as a special case of the loss network having only a single trunk; then the circuits on the trunk constitute the resource to be shared. This first version of the resource-sharing model can be thought of as an *infinite-server variant*, because each request enters service immediately upon arrival (unless it is blocked) and has a sojourn time that does not depend on the other requests present. Another version is a *single-server variant,* as in a memory buffer at a node in a packet-switching network in which each class is served by one outgoing link. Then the packets occupy some buffers while waiting to be transmitted, as in Kamoun and Kleinrock (1980). Even if a processor-sharing discipline is used, so that requests enter service immediately upon arrival, the sojourn time depends strongly on the other requests that are present. In this paper, we consider only the infinite-server variant of the resource-sharing model. We intend to discuss the single-server variant and more general many-server-variants in subsequent papers. Early work on the infinite-server variant was motivated by sharing bandwidth on a satellite channel (e.g., Aein and Kosovych (1977) and Aein (1978)). Mitra and Morrison (1993) discuss important new applications of resource-sharing models to integrated multi-rate services on ATM and wireless networks.

The standard resource-sharing model has requests arriving according to independent Poisson processes, one for each class of requests. However, it is also possible to consider variants of the resource-sharing model with state-dependent arrival rates, as was done in the case of linear arrived rates (the binomial-Poisson-Pascal model) by Delbrouck (1983), Dziong and Roberts (1987), and Mitra and Morrison (1993). Here we confine attention to Poisson arrivals, but we extend the inversion algorithm to state-dependent arrivals in Choudhury, Leung and Whitt

(1994a).

Recursive algorithms for computing normalization constants in resource-sharing models were first developed by Kaufman (1981) and Roberts (1981) for the case of complete sharing (CS) with Poisson arrivals. A recursive algorithm for the CS case with state-dependent arrivals (the BPP model) was then developed by Delbrouck (1983). Delbrouck's algorithm was extended to multiple resources by Dziong and Roberts (1987). A new recursion for the multi-resource BPP model has recently been developed by van de Vlag and Awater (1994). Further recursions were developed for the case of batch Poisson arrivals with CS by Kaufman and Rege (1993) and van Doorn and Panken (1993). (Our methods also apply to batch arrivals; we discuss both batch Poisson arrivals and state-dependent batch arrivals in Choudhury, Leung and Whitt (1994b).) A recursive algorithm for the UL sharing policy has recently been developed by Chuah (1993). These recursions are all similar in spirit to Buzen's (1973) classical convolution algorithm for closed queueing networks.

These recursive algorithms are effective when the model is not too large, but they tend to become ineffective as the model grows, due to numerical underflow/overflow, slowness or roundoff error. For large models, we can use asymptotic approximations, as in Evans (1991), Reiman (1991), Labourdette and Hart (1992), and Mitra and Morrison (1993). The *uniform asymptotic approximation* (UAA) of Mitra and Morrison seem especially effective for analyzing large models, but even UAA can experience difficulties with large models if not all parameters are suitably large, as we show in Choudhury, Leung and Whitt (1994a).

For resource sharing models, our main contributions are to consider multiple resources and develop an effective algorithm for non-CS policies as well as the CS policy. Previous algorithms have primarily been for a single resource with the CS policy. However, even in the relatively familiar setting of a single resource with the CS policy, our algorithm can treat cases not covered by any previous algorithm, as we show in Section 7.

1.3 The Numerical Inversion Algorithm

Our algorithm follows the approach in our previous paper, Choudhury, Leung and Whitt (1993). In that paper we developed an algorithm for calculating normalization constants and moments in product-form models by numerically inverting their generating functions. For that purpose, we use the numerical inversion algorithm developed by Abate and Whitt (1992a,b) and enhanced by Choudhury, Lucantoni and Whitt (1994), which is based on the Fourier-series method. That algorithm (reviewed in Section 3) requires that we can indeed obtain values of the generating function and that we can perform a suitable scaling to control the errors.

In Choudhury, Leung and Whitt (1993) we developed a full algorithm for a class of closed queueing networks. This was a convenient first class of product-form models to consider, because it was known that the generating function of the normalization constant can be expressed in a closed, compact form; see Reiser and Kobayashi (1975) and Bertozzi and McKenna (1993). Here we develop a full version of the numerical inversion algorithm for the multi-rate multi-class multi-resource *loss models* described above. Our first contribution here is to derive the generating functions of the normalization constants for these models (Section 2). We do this for all three sharing policies; CS, UL and GM. It is significant that these generating functions also can be expressed in a remarkably simple closed, compact form. There has been less previous work with generating functions of normalization constants for loss models than for closed queueing networks. Generating functions of normalization constants in some loss models were previously derived by Delbrouck (1983), Mitra (1987), Mitra and Morrison (1993) and Morrison (1993). Their derivations are less general since they involve only the CS policy, and they have simpler network structures. (However, Morrison (1993) consider the more general batch-Poisson arrivals.)

Our second contribution here is to develop an effective static scaling algorithm to control the discretization error (Section 4). As with the closed queueing networks, the scaling is an essential step since the normalization constant can be very large or very small. The general principles for developing effective scaling apply as before, but we cannot just apply the old scaling algorithm. Here we develop a new scaling algorithm especially tailored to the loss models.

While the numerical inversion algorithm is very effective for treating certain kinds of largeness, e.g., large capacity in a resource and many request classes, the computational complexity grows exponentially with the number of resources. (See Section 5.) Hence, we are not nearly able to analyze a complex product-form loss model of the size of the AT&T long-distance network. Nevertheless, just as with the closed queueing networks, fortunately the computation can often be reduced dramatically by exploiting conditional decomposition based on special structure (Section 3.1). For example, the dimension can be reduced to 2 for the two-hop tree networks considered by Mitra (1987), Kogan (1989) and Tsang and Ross (1990) even though the number of trunks can be arbitrarily large. Other structured models exhibit similar dramatic dimension reduction by conditional decomposition.

For the loss models, we find that the computation can often be further reduced by judicious truncation of large finite sums (Section 5). For example, we show that for a single resource model with 100,000 resource units the computational savings can be by a factor of 200 with essentially no loss of accuracy. For multiple resources, similar savings can be obtained in each dimension, so that the overall savings grows geometrically with the number of resources.

1.4 Validation of the Inversion Algorithm

In this paper we also develop an alternative direct algorithm in order to validate our numerical inversion algorithm on small models (Section 6). Interestingly, the direct algorithm has regions where it is more effective than the inversion algorithm. Hard problems for the direct algorithm tend to be duals of the hard problems for the inversion algorithm, with duality meaning that we switch the role of classes and resources. The inversion (direct) algorithm thus tends to perform well when there are only relatively few resources (classes), but there can be many classes (resources). Furthermore, we can also exploit conditional decomposition based on special structure for the direct algorithm, so that it can perform well with very large numbers of classes.

Finally, we consider four classes of numerical examples (Sections 7-10). We start by considering the classical single-resource resource-sharing model with the CS policy (Section 7). Since there is only one resource, the inversion is only one dimensional. Thus this example is very easy for our algorithm. In addition to making comparisons with the direct algorithm for problems that have few classes, for this model we also implemented the Kaufman (1981) – Roberts (1981) recursion and the Mitra and Morrison (1993) uniform asymptotic approximation (UAA). We verify the validity and effectiveness of our algorithm by showing that our algorithm agrees with the recursion for small models and agrees with Mitra and Morrison's UAA for large models. Our algorithm also has a self-contained accuracy check. We verify accuracy by performing calculations with different inversion parameters (corresponding to different contours on the inversion integrals). Since the computations are indeed different, the accuracy is indicated by the extent of the agreement.

We also consider variations of the resource-sharing model with the UL and GM sharing policies as well as CS. We show that our algorithm is able to treat these other sharing policies just as effectively as CS (Section 8). For a single resource, the dimension of the inversion always can be reduced to at most two. Hence, these variations are not difficult for our algorithm.

To illustrate our algorithm for loss networks, we consider the multi-rate extension of the 5-resource example in Figure 1, taken from the introduction of Kelly (1986) (Section 9) and networks with special structure, such as tree networks (Section 10). Even though the Kelly example with only five trunks is relatively small by communication network standards, it is already a challenging model to analyze. Analysis by our algorithm is facilitated by reducing the dimension from five to three by exploiting conditional decomposition. For tree networks, we consider both UL and CS sharing policies, thus going beyond previous algorithms. When no alternative algorithm is available, we rely on our self-contained accuracy check. Readers could go next to the numerical examples in Sections 7-10.

2. The Generating Functions

In this section we derive the generating functions of the normalization constants. In Subsection 2.1 we review the general product-form solution for the loss model with Poisson arrivals. In Subsection 2.2 we indicate how to treat other steady-state characteristics besides blocking probabilities, in particular, factorial moments of marginal distributions. In Subsections 2.3, 2.4 and 2.5 we consider this loss model with the CS, UL and GM sharing policies, respectively. In Subsection 2.6 we consider the same model with mixed sharing policies, where the three sharing policies are each used by subsets of the requests (on all resources).

To emphasize the wide applicability of the models, we use the generic resource-sharing terminology of resources, requests and capacity (or units) instead of trunks, calls and circuits, respectively.

2.1 The General Product-Form Solution

Consider a loss model with *p resources* and *r classes or types of requests.* Let the resources be indexed by i and the type of request by j. Let resource i have *capacity* K_i, $1 \le i \le p$, 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 type j request requires a_{ij} units of 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 requests come from independent Poisson processes, with requests of type j having *arrival rate* λ_j. Let the holding times be mutually independent and independent of the arrival processes. Let the *mean holding time* of a type j request be t_j. Thus the *offered load* of type j is $\rho_j \equiv \lambda_j t_j$. Each request is accepted if all desired resources can be provided; otherwise, the request is blocked and lost. All resources used by a request are released at the end of the holding time.

Let the system *state vector* be $\mathbf{n} \equiv (n_1, \ldots, n_r)$, where n_j is the number of type j requests currently being satisfied. Let $S_P(\mathbf{K})$ be the set of allowable states, which depends on the capacity vector $\mathbf{K}$ and the *sharing policy P*. With non-CS policies, the set of allowable states will typically depend on other parameters besides $\mathbf{K}$.

If the holding times are exponentially distributed, then the stochastic process $\{\mathbf{N}(t) : t \ge 0\}$, where $\mathbf{N}(t)$ gives the system state at time t, is a finite-state continuous-time Markov chain (CTMC). We will only consider cases in which this CTMC is irreducible, and we will be interested in its unique steady-state probability mass function π. Through insensitivity, this steady-state distribution also holds for non-exponential holding times.

The steady-state probability mass function has the simple *product form*

$$\pi(\mathbf{n}) = g(\mathbf{K})^{-1} \prod_{j=1}^{r} \frac{\rho_j^{n_j}}{n_j!} , \tag{2.1}$$

with the *normalization constant*

$$g(\mathbf{K}) \equiv g_P(\mathbf{K}) = \sum_{\mathbf{n} \in S_P(\mathbf{K})} \prod_{j=1}^{r} \frac{\rho_j^{n_j}}{n_j!} . \tag{2.2}$$

Formulas (2.1) and (2.2) are consequences of the product-form model theory; e.g., see Kelly (1979). Formula (2.1) is easily derived by initially considering the infinite-server model without any capacity constraints. In the case of exponential holding times, the Markov process $\mathbf{N}(t)$ is easily seen to be irreducible positive recurrent and reversible (corresponding to independent $M/M/\infty$ queues) with a stationary distribution that is the product of independent Poisson distributions, i.e.,

$$\pi(\mathbf{n}) = \prod_{j=1}^{r} e^{-\rho_j} \frac{\rho_j^{n_j}}{n_j!} . \tag{2.3}$$

By Section 1.6 of Kelly (1979), the steady-state distribution with capacity constraints is just the truncation of (2.3) to a smaller state space with a renormalization to yield total mass 1; obviously this truncation and renormalization of (2.3) is just (2.1). Alternatively we can just check that (2.1) satisfies the global balance conditions for π to be the steady-state distribution of the CTMC $\mathbf{N}(t)$. Since Poisson arrivals see time averages, e.g., Melamed and Whitt (1990), the steady-state distribution seen by an arrival is also given by (2.1).

Basic properties of the Poisson distribution help us see that the probability mass function $\pi(\mathbf{n})$ in (2.1) is well behaved. For example, $\rho^n / n!$ is unimodal in n with a maximum at either $\lfloor \rho \rfloor$ or $\lfloor \rho \rfloor + 1$, where $\lfloor x \rfloor$ is the greatest integer less than or equal to x. This structure helps us develop effective scaling in Section 4.

We calculate the normalization constants by numerically inverting the generating function

$$G(\mathbf{z}) \equiv \sum_{K_1=0}^{\infty} \sum_{K_2=0}^{\infty} \cdots \sum_{K_p=0}^{\infty} g(\mathbf{K}) z_1^{K_1} z_2^{K_2} \cdots z_p^{K_p} , \tag{2.4}$$

where $\mathbf{z} \equiv (z_1, z_2, \ldots, z_p)$ is a vector of complex variables. (For the UL and GM sharing policies, there will be extra variables in g and thus G.)

From (2.2) it is evident that $g(\mathbf{K})$ is increasing in $\mathbf{K}$,

$$g(\mathbf{K}) \le e^{\rho_1 + \ldots + \rho_r} \text{ for all } \mathbf{K}$$

and $g(\mathbf{K})$ approaches this upper bound as $K_j \to \infty$ for all j. Hence, the generating function $G(\mathbf{z})$ is absolutely convergent, and thus analytic, in the open polydisc $\{\mathbf{z} : |\mathbf{z}_j| < 1,\ 1 \le j \le p\}$. For numerical inversion, the key point is that the generating function $G(\mathbf{z})$ can be expressed conveniently in a closed, compact form.

A generic expression for the blocking probability for class j is

$$B_j \; = \; 1 \; - \; \frac{\displaystyle\sum_{n \in S'_p(K)} \prod_{j=1}^{r} \frac{\rho_j^{n_j}}{n_j\,!}}{\displaystyle\sum_{n \in S_p(K)} \prod_{j=1}^{r} \frac{\rho_j^{n_j}}{n_j\,!}} \;, \tag{2.5}$$

where $S_p(K)$ is defined in (2.2) and $S'_p(K)$ is the subset of states in $S_p(K)$ in which another class j request can be accepted. The denominator of the second term in (2.5) is the standard normalization constant, while the numerator can be expressed as another normalization constant.

2.2 Steady-State Quantities Other than Blocking Probabilities

In this paper we only consider blocking probabilities, but other steady-state quantities such as marginal distributions and moments can also be expressed in terms of normalization constants, usually involving only two normalization constant values. For example, let N_j be the steady-state number of class j in service with the CS policy. It is easy to see that the marginal distribution is given by

$$P(N_j \; = \; n_j) \; = \; \frac{\rho_j^{n_j}}{n_j\,!} \; \frac{g^{(j)}(\mathbf{K} - \mathbf{A}\mathbf{e}_j n_j)}{g(\mathbf{K})} \;, \tag{2.6}$$

where $g^{(j)}(\mathbf{K})$ represents the normalization constant of the system with class j removed.

We can treat moments as indicated in Section 6 of Choudhury, Leung and Whitt (1993). Since it is convenient to treat factorial moments, let

$$M_{jk} \; = \; \sum_{n_j = k}^{\infty} n_j (n_j - 1) \ldots (n_j - k + 1) P(N_j = n_j) \; . \tag{2.7}$$

By (2.6),

$$M_{jk} \; = \; \sum_{n_j = k}^{\infty} n_j (n_j - 1) \ldots (n_j - k + 1) \frac{\rho_j^{n_j}}{n_j\,!} \; \frac{g^{(j)}(\mathbf{K} - \mathbf{A}\mathbf{e}_j n_j)}{g(\mathbf{K})} \; .$$

Let $\bar{n}_j \; = \; n_j - k$. Then,

$$M_{jk} \; = \; \rho_j^k \sum_{\bar{n}_j \; = \; 0}^{\infty} \frac{\rho_j^{\bar{n}_j}}{\bar{n}_j\,!} \; \frac{g^{(j)}(\mathbf{K} - \mathbf{A}\mathbf{e}_j k - \mathbf{A}\mathbf{e}_j \bar{n}_j)}{g(\mathbf{K})} \; = \; \rho_j^k \; \frac{g(\mathbf{K} - \mathbf{A}\mathbf{e}_j k)}{g(\mathbf{K})} \; . \tag{2.8}$$

So all the factorial moments may be computed just as easily as the blocking probability, namely, by computing the normalization constant at two values. The mean is obtained as

$$M_{j1} \; = \; \rho_j \frac{g(\mathbf{K} - \mathbf{A}\mathbf{e}_j)}{g(\mathbf{K})} \; . \tag{2.9}$$

Using (2.9) and (2.11) below, we get $M_{j1} \; = \; \lambda_j (1 - B_j) \mu_j^{-1}$, which agrees with what we get by applying Little's law.

2.3 Complete Sharing

With complete sharing (CS), the only capacity constraint is provided by the capacity vector $\mathbf{K}$. The allowable states are now the set of state vectors $\mathbf{n}$ such that $\sum_{j=1}^{r} a_{ij} n_j \le K_i$, $1 \le i \le p$; i.e.,

$$S_{CS}(\mathbf{K}) = \{ \mathbf{n} \in \mathbf{Z}_+^r : \mathbf{A}\mathbf{n} \le \mathbf{K} \} , \tag{2.10}$$

where $\mathbf{Z}_+$ is the set of nonnegative integers and $\mathbf{Z}_+^r$ is its r-fold cartesian product. From (2.10) we see that with the CS policy each resource produces a linear constraint; i.e., resource i produces the constraint $\sum_{j=1}^{r} a_{ij} n_j \le K_i$ where a_{ij}, n_j and K_i are integers. Thus additional linear constraints are equivalent to introducing additional resources. To obtain integer parameters, we assume that the coefficients a_{ij} are all rational; then there is an equivalent representation in which a_{ij} and K_i are integers.

By (2.5), the stationary blocking probability of an arbitrary type j request is given by

$$B_j = 1 - \frac{g(\mathbf{K} - \mathbf{A}\mathbf{e}_j)}{g(\mathbf{K})} , \tag{2.11}$$

where $\mathbf{e}_j$ is the j^{th} r-dimensional *unit vector* (with a 1 in the j^{th} place and 0's elsewhere), which here we regard as a column vector. We calculate B_j in (2.11) by calculating $g(\mathbf{K} - \mathbf{A}\mathbf{e}_j)$ and $g(\mathbf{K})$.

The key to obtain a convenient closed-form expression for the generating function is changing the order of summation from the given order in (2.2) and (2.4). Doing so for $G(\mathbf{z})$, we obtain

$$G(\mathbf{z}) = \sum_{n_1=0}^{\infty} \cdots \sum_{n_r=0}^{\infty} \sum_{K_1 = \sum_{j=1}^{r} a_{1j} n_j}^{\infty} \cdots \sum_{K_p = \sum_{j=1}^{r} a_{pj} n_j}^{\infty} \prod_{j=1}^{r} \frac{\rho_j^{n_j}}{n_j!} z_1^{K_1} \cdots z_p^{K_p} = \frac{\exp\left[\sum_{j=1}^{r} \rho_j \prod_{i=1}^{p} z_i^{a_{ij}} \right]}{\prod_{i=1}^{p} (1 - z_i)} \tag{2.12}$$

2.4 Sharing with Upper Limits

We have indicated above how to treat additional linear constraints on the state space by adding additional resources. Now we consider two special sharing policies for which we can reduce the dimension of the generating function. We now assume that, in addition to the overall capacity constraint provided by $\mathbf{K}$, an upper limit L_{ij} is imposed on request j for resource i for each (i, j) pair. At first glance, this appears to be rp new constraints, but it is easy to see that there are at most r binding constraints, one for each type of request. We first identify the resource that limits the maximum number of type j requests to be carried by the network simultaneously. Resource i is the *type-j limiting resource* if

$$\lfloor L_{ij} / a_{ij} \rfloor \le \lfloor L_{i'j} / a_{i'j} \rfloor \quad \text{for all } 1 \le i' \le p . \tag{2.13}$$

If resource i is the type-j limiting resource, then we let $M_j = \lfloor L_{ij} / a_{ij} \rfloor$. Note that

$$\{\mathbf{n} \in \mathbf{Z}_+^r : a_{ij} n_j \le L_{ij} : 1 \le i \le p\ ,\ 1 \le j \le r\} = \{\mathbf{n} \in \mathbf{Z}_+^r : n_j \le M_j\ ,\ 1 \le j \le r\}\ .$$

Introduce the vector $\mathbf{M} \equiv (M_1, \ldots, M_r)$ and note that the set of feasible states with UL is

$$S_{UL}(\mathbf{K},\mathbf{M}) = \{\mathbf{n} \in \mathbf{Z}_+^r : \mathbf{An} \le \mathbf{K}\ ,\ \mathbf{n} \le \mathbf{M}\}\ . \qquad ((2.14)$$

Let $g(\mathbf{K},\mathbf{M})$ be the normalization constant as a function of the pair $(\mathbf{K},\mathbf{M})$. Paralleling (2.11), the blocking probability is now

$$B_j = 1 - \frac{g(\mathbf{K} - \mathbf{A}\mathbf{e}_j, \mathbf{M} - \mathbf{e}_j)}{g(\mathbf{K},\mathbf{M})}\ . \qquad (2.15)$$

Since $(\mathbf{K},\mathbf{M})$ is of dimension $p + r$, so will be the generating function of $g(\mathbf{K},\mathbf{M})$ (but in Section 3.1 we will show that the dimension always can be reduced to $p+1$ or p for the numerical inversion). Let $\mathbf{y} \equiv (y_1, \ldots, y_r)$ be a vector of complex variables which we will use for the last r dimensions. Thus, we can define the generating function of $g(\mathbf{K},\mathbf{M})$ as

$$G(\mathbf{z},\mathbf{y}) = \sum_{K_1=0}^{\infty} \sum_{K_2=0}^{\infty} \cdots \sum_{K_p=0}^{\infty} \sum_{M_1=0}^{\infty} \sum_{M_2=0}^{\infty} \cdots \sum_{M_r=0}^{\infty} \sum_{\mathbf{n} \in S_{UL}(\mathbf{K},\mathbf{M})}$$

$$\prod_{j=1}^{r} \frac{\rho_j^{n_j}}{n_j!}\ z_1^{K_1} z_2^{K_2} \cdots z_p^{K_p} y_1^{M_1} y_2^{M_2} \cdots y_r^{M_r}\ . \qquad (2.16)$$

We change the order of summation, just as we did for the CS policy, in order to obtain a closed-form expression:

$$G(\mathbf{z},\mathbf{y}) = \sum_{n_1=0}^{\infty} \cdots \sum_{n_r=0}^{\infty} \sum_{M_1=n_1}^{\infty} \cdots \sum_{M_r=n_r}^{\infty}$$

$$\sum_{K_1=\sum_{j=1}^{r} a_{1j} n_j}^{\infty} \cdots \sum_{K_p=\sum_{j=1}^{r} a_{pj} n_j}^{\infty} \prod_{j=1}^{r} \frac{\rho_j^{n_j}}{n_j!} z_1^{K_1} \cdots z_p^{K_p} y_1^{M_1} \cdots y_r^{M_r}$$

$$= \frac{\exp\left[\sum_{j=1}^{r} \rho_j y_j \prod_{i=1}^{p} z_i^{a_{ij}}\right]}{\prod_{i=1}^{p} (1 - z_i) \prod_{j=1}^{r} (1 - y_j)}\ . \qquad (2.17)$$

Looking at (2.17) and (2.12), we see that a loss model with p resources, r classes and the UL policy is equivalent to a special case of a loss model having $p + r$ resources, r classes and the CS policy: just let $z_{p+j} = y_j$ for $1 \le j \le r$. The capacity limit on resource $p + j$ is the upper limit M_j for class j and a class j request requires a_{ij} resource units from resource i, $1 \le i \le p$, 1 resource unit from resource $p + j$, and no resource units from resource $p + k$ for $k \ne j$ and $1 \le k \le r$.

2.5 Sharing with Guaranteed Minima

With the GM policy, a certain number of units in each resource are reserved for each type of request. It is easy to see that UL and GM policies are equivalent for two classes, but not for more than two. For example, with three classes, GM puts upper limits on the total amounts of the resource used by pairs of the classes. More generally, the GM policy is equivalent to introducing constraints for each resource on the weighted sum of the numbers of requests from all but one class. Thus, the GM policy also corresponds to a set of linear constraints, so that it could be implemented by just adding resources. However, we will show how to exploit the special structure to reduce the dimension.

To obtain a tractable expression in this case we assume that the requirements matrix **A** has a special form. In particular, let there be a vector $\mathbf{b} \equiv (b_1, \ldots, b_r)$ such that a_{ij} equals either b_j or 0 for all i and j. Moreover, we assume that the amount of capacity guaranteed for each type of request is the same for all resources used by that type of request. Let M_j be the capacity guaranteed to type j request in each of its required resources and let **M** denote the vector $(M_1, \ldots, M_r)$. Note that M denotes different quantities for the UL and GM policies.

For the i^{th} resource with capacity K_i, $M_j \delta_{ij}$ resource units are reserved for type j requests, $j = 1,2,\ldots r$ where $\delta_{ij} = 1$ if $a_{ij} > 0$ and $\delta_{ij} = 0$ otherwise. So $K_i - \sum_{j=1}^{r} M_j \delta_{ij}$ resource units are commonly shared. If $a_{ij} > 0$, then type j requests can use any of the M_j resource units reserved for them and any of the commonly shared resource units. A new type-j request is blocked if the type-j requests already occupy more than the guaranteed minimum on some resource and if not enough resource units are available at that resource. Given **K** and **M**, let $S_{GM}(\mathbf{K},\mathbf{M})$ be the set of all allowable system states under the GM policy. Then

$$S_{GM}(\mathbf{K},\mathbf{M}) = \{\mathbf{n} \in \mathbf{Z}_+^r : \sum_{j=1}^{r} \max(a_{ij} n_j, \delta_{ij} M_j) \le K_i \text{ for } i = 1, \ldots, p\} \ . \qquad (2.18)$$

Let $g(\mathbf{K},\mathbf{M})$ be the normalization constant. Paralleling (2.11) and (2.15), the blocking probability now, by (2.5), is

$$B_j = 1 - \frac{g(\mathbf{K} - \mathbf{A}\mathbf{e}_j, \mathbf{M} - \mathbf{A}\mathbf{e}_j)}{g(\mathbf{K},\mathbf{M})} \ . \qquad (2.19)$$

As for the CS and UL policies, the order of summation can be changed to obtain a close-form expression for the generating function of $g(K,M)$. For a given system state **n**, observe that each M_j has no effect in causing **n** to be valid or not (for some sufficiently large capacity vector **K**). Consequently, for the given **n**, M_j can vary from zero to infinity. Next, consider the capacity K_i. For n_j type j requests, the number of units in resource i occupied or reserved for type-j requests is the maximum of $a_{ij} n_j$ and $\delta_{ij} M_j$. To account for all request types, K_i must be at least $\sum_{j=1}^{r} \max(a_{ij} n_j, \delta_{ij} M_j)$. Thus we can write the generating function as

$$G(\mathbf{z},\mathbf{y}) = \sum_{n_1=0}^{\infty} \cdots \sum_{n_r=0}^{\infty} \sum_{M_1=0}^{\infty} \cdots \sum_{M_r=0}^{\infty} \sum_{K_1 = K_1(\mathbf{n},\mathbf{M})}^{\infty} \cdots \sum_{K_p = K_p(\mathbf{n},\mathbf{M})}^{\infty} \prod_{j=1}^{r} \frac{\rho_j^{n_j}}{n_j!} y_1^{M_1} \cdots y_r^{M_r} z_1^{K_1} \cdots z_p^{K_p} \ ,$$

where $K_i(\mathbf{n},\mathbf{M}) = \sum_{j=1}^{r} \max\{a_{ij} n_j, \delta_{ij} M_j\}$. Carrying out the summation over all K_i, then over all the M_j (breaking the sum over M_j into two parts, below and above $b_j n_j$), and finally over all n_j we obtain

$$G(\mathbf{z},\mathbf{y}) = \left[\prod_{i=1}^{p} \frac{1}{1-z_i} \right] \times$$

$$\prod_{j=1}^{r} \left\{ \frac{\exp(\rho_j \prod_{i=1}^{p} z_i^{a_{ij}}) - y_j \exp(\rho_j y_j^{b_j} \prod_{i=1}^{p} z_i^{a_{ij}})}{1-y_j} + \frac{y_j \prod_{i=1}^{p} z_i^{\delta_{ij}} \exp(\rho_j y_j^{b_j} \prod_{i=1}^{p} z_i^{\delta_{ij} b_j})}{1-y_j \prod_{i=1}^{p} z_i^{\delta_{ij}}} \right\} . \quad (2.20)$$

As with the UL policy, the dimension of the generating function with the GM policy is $p+r$, but in Section 3.1 we will show for the numerical inversion that the dimension can always be reduced to $p + 1$ or p.

2.6 Mixed Sharing Policies

We now consider a multi-resource loss model with r types of requests, where different requests use different sharing policies. For a given class, we assume that the same sharing policy is used on all requested resources. Thus there are integers r_1 and r_2 with $1 \le r_1 \le r_2 \le r$ such that classes $1, \ldots, r_1$ use the CS policy, classes $r_1+1,\ r_1+2, \ldots, r_2$ use the UL policy and classes $r_2+1,\ r_2+2, \ldots, r$ use the GM policy.

Just as in Section 2.5, we place restrictions on the requirements per request a_{ij} for requests using the GM policy. In particular, a_{ij} is either b_j or 0 for all i. Similarly, we require that the capacity guarantee M_j for class j is the same on all resources used by class j. We continue to use M_j to refer to both the upper limits for requests with the UL policy and guaranteed minimum number of resource units for the GM policy.

To identify all request types appropriately, let $\mathbf{n} \equiv (n_1, \ldots, n_{r_1}, n_{r_1+1}, \ldots, n_{r_2}, n_{r_2+1}, \ldots, n_r)$, $\mathbf{M} \equiv (M_{r_1+1}, \ldots, M_{r_2}, M_{r_2+1}, \ldots, M_r)$ and $\mathbf{y} \equiv (y_{r_1+1}, \ldots, y_{r_2}, y_{r_2+1}, \ldots, y_r)$. We use $S_{MP}(\mathbf{K},\mathbf{M})$ to denote all feasible system states under these mixed policies, which is

$$S_{MP}(\mathbf{K},\mathbf{M}) = \{ \mathbf{n} \in \mathbf{Z}_+^r : n_j \le M_j \text{ for } j = r_1+1, r_1+2, \ldots, r_2$$

$$\text{and } \sum_{j=1}^{r_2} a_{ij} n_j + \sum_{j=r_2+1}^{r} \max(a_{ij} n_j, \delta_{ij} M_j) \le K_i \text{ for } i = 1, \ldots, p \} .$$

The conditions for being feasible states simply reflect that the upper limits have to be met for the UL policy, the guaranteed minima are satisfied for the GM policy, and that enough capacity is available in each resource for requests using all three sharing policies. By the definition of the generating function,

$$G(\mathbf{z},\mathbf{y}) \;=\; \sum_{M_{r_1+1}=0}^{\infty} \; \sum_{M_{r_1+2}=0}^{\infty} \cdots \sum_{M_r=0}^{\infty} \; \sum_{K_1=0}^{\infty} \; \sum_{K_2=0}^{\infty} \cdots \sum_{K_p=0}^{\infty} \; \sum_{\mathbf{n}\in S_{MP}(\mathbf{K},\mathbf{M})}$$

$$\prod_{j=1}^{r} \frac{\rho_j^{n_j}}{n_j!} \; z_1^{K_1} z_2^{K_2} \cdots z_p^{K_p} y_{r_1+1}^{M_{r_1+1}} y_{r_1+2}^{M_{r_1+2}} \cdots y_r^{M_r} \; .$$

Once again, after changing the order of summation, we obtain

$$G(\mathbf{z},\mathbf{y}) = \left[\prod_{i=1}^{p} \frac{1}{1-z_i} \right] \exp\left[\sum_{j=1}^{r_1} \rho_j \prod_{i=1}^{p} z_i^{a_{ij}} \right] \exp\left[\sum_{j=r_1+1}^{r_2} \rho_j y_j \prod_{i=1}^{p} z_i^{a_{ij}} \right] \prod_{j=r_1+1}^{r_2} \left[\frac{1}{1-y_j} \right]$$

$$\prod_{j=r_2+1}^{r} \left\{ \frac{\exp(\rho_j \prod_{i=1}^{p} z_i^{a_{ij}}) - y_j \exp(\rho_j y_j^{b_j} \prod_{i=1}^{p} z_i^{a_{ij}})}{1-y_j} + \frac{y_j \prod_{i=1}^{p} z_i^{\delta_{ij}} \exp(\rho_j y_j^{b_j} \prod_{i=1}^{p} z_i^{\delta_{ij} b_j})}{1-y_j \prod_{i=1}^{p} z_i^{\delta_{ij}}} \right\} \qquad (2.21)$$

Clearly, the generating functions in (2.12), (2.17) and (2.20) are special cases of (2.21), in which only one of the sharing policies is used for all request types throughout the model. Dimension reduction applies to the mixed policies just as it does to the UL and GM policies; see Section 3.1.

3. The Numerical Inversion Algorithm

In this section we briefly review the numerical inversion algorithm; for background and more details see Abate and Whitt (1992a,b), Choudhury, Lucantoni and Whitt (1994) and Choudhury, Leung and Whitt (1993). In Subsection 3.1 we discuss dimension reduction by conditional decomposition, and in Subsection 3.2 we review the basic algorithm. We develop our scaling algorithm for the loss models in Section 4 and discuss other algorithmic issues in Section 5. Throughout the next three sections we use the notation in (2.4) and (2.12); thus our goal is to calculate $g(\mathbf{K})$ given $G(\mathbf{z})$ where $\mathbf{K}$ and $\mathbf{z}$ are p-dimensional.

3.1 Dimension Reduction

Our approach to inverting p-dimensional generating functions is to recursively perform p one-dimensional inversions. In general, the computational complexity is exponential in the dimension p, but a dramatic reduction in computation often occurs due to special structure if we perform the one-dimensional inversions in a good order.

We look for *conditional decomposition.* We select d variables which we are committed to invert. We then look at the generating function with these d variables fixed, and we write the function of the remaining $p-d$ variables as a product of factors, where no two factors have any variables in common. Since factors without any common variables can be treated separately, the maximum dimension of the additional inversion required beyond the designated d variables is equal to the maximum number of the $p-d$ remaining variables appearing in one of the factors, say m. The overall inversion can then be regarded as being of dimension $d + m$. The idea, then, is to select an appropriate d variables, so that the resulting dimension $d + m$ is as small as possible. A systematic procedure is presented in Choudhury, Leung and Whitt (1993).

We note that recursive algorithms exploiting forms of decomposition have been developed previously. The first evidently was the tree convolution algorithm of Lam and Lien (1983); it applies to closed queueing networks. A recursive algorithm for loss networks exploiting decomposition has been developed by Conway and Pinsky (1992). Our algorithm is very different from these recursive algorithms.

From (2.12), note that indeed $G(\mathbf{z})$ can be written as a product of factors. In the denominator a single variable z_i appears in each factor. The exponential term can be written as a product of factors with one factor for each class. The variable z_i appears in the exponential factor for class *j* if and only if class *j* requires resource *i*, i.e., if $a_{ij} > 0$.

The generating function for the UL policy in (2.17) appears to require a $(p + r)$-dimensional inversion, but exploiting special structure, the problem always can be reduced to a $(p + 1)$-dimensional or even an *p*-dimensional inversion. If we at first keep all the z_i variables fixed, then the inversions for the different y_j variables clearly can be done separately. This reduces the dimension to $p + 1$. Furthermore, it is possible to invert with respect to each y_j variable *explicitly* to get

$$G(\mathbf{z},\mathbf{M}) = \frac{1}{\prod_{i=1}^{p}(1-z_i)} \prod_{j=1}^{r} \sum_{l=0}^{M_j} \frac{(\rho_j \prod_{i=1}^{p} z_i^{a_{ij}})^l}{l!} , \tag{3.1}$$

which clearly requires a *p*-dimensional inversion. If M_j is very large, we recommend using the $(p + 1)$-dimensional inversion to take advantage of truncation; otherwise use (3.1).

Similarly, from (2.20) the generating function for the GM policy appears to be $(p + r)$-dimensional, but it too always can be reduced to $(p + 1)$-dimensional or *p*-dimensional. As for the UL policy, the inversion with respect to y_j may be done explicitly for the GM policy as well, yielding

$$G(\mathbf{z},\mathbf{M}) = \frac{1}{\prod_{i=1}^{p}(1-z_i)} \prod_{j=1}^{r} \left[\exp(\rho_j \prod_{i=1}^{p} z_i^{a_{ij}}) - \sum_{l=0}^{\left\lfloor \frac{M_j-1}{b_j} \right\rfloor} \frac{(\rho_j \prod_{i=1}^{p} z_i^{a_{ij}})^l}{l!} \right.$$

$$\left. + \, (\prod_{i=1}^{p} z_i^{\delta_{ij}})^{M_j} \sum_{l=0}^{\left\lfloor \frac{M_j-1}{b_j} \right\rfloor} \frac{\rho_j^l}{l!} \right] . \tag{3.2}$$

A similar reduction in dimension from $p + r$ to $p + 1$ or p is possible for the mixed sharing policy in Section 2.6 as well.

3.2 The Basic Algorithm

Given a *p*-dimensional generating function $G(\mathbf{z})$, we first do the dimension reduction analysis to determine the order of the variables to be inverted. Given that the order has been specified, we perform (up to) *p* one-dimensional inversions recursively.

To represent the recursive inversion, we define *partial generating functions* by

$$g^{(j)}(\mathbf{z}_j,\mathbf{K}_{j+1}) = \sum_{K_1=0}^{\infty} \cdots \sum_{K_j=0}^{\infty} g(\mathbf{K}) \prod_{i=1}^{j} z_i^{K_i} \quad \text{for } 0 \le j \le p\ , \tag{3.3}$$

where $\mathbf{z}_j = (z_1, z_2, \ldots, z_j)$ and $\mathbf{K}_j = (K_j, K_{j+1}, \ldots, K_p)$ for $1 \le j \le p$. Let $\mathbf{z}_0$ and $\mathbf{K}_{p+1}$ be null vectors. Clearly, $\mathbf{K} = \mathbf{K}_1, \mathbf{z} = \mathbf{z}_p$, $g^{(p)}(\mathbf{z}_p, \mathbf{K}_{p+1}) = G(\mathbf{z})$ and $g^{(0)}(\mathbf{z}_0, \mathbf{K}_1) = g(\mathbf{K})$.

Let I_j represent inversion with respect to z_j. Then the step-by-step nested inversion approach is

$$g^{(j-1)}(\mathbf{z}_{j-1},\mathbf{K}_j) = I_j[g^{(j)}(\mathbf{z}_j, \mathbf{K}_{j+1})]\ , \quad 1 \le j \le p\ , \tag{3.4}$$

starting with $j = p$ and decreasing j by 1 each step. In the actual program implementation, we attempt the inversion shown in (3.4) for $j = 1$. In order to compute the righthand side we need another inversion with $j = 2$. This process goes on until at step p the function on the righthand side becomes the p-dimensional generating function and is explicitly computable.

In each step we use the LATTICE-POISSON inversion algorithm in Abate and Whitt (1992a,b) with modifications to improve precision and allow for complex inverse functions as in Choudhury, Lucantoni and Whitt (1994a). We show below the inversion formula at the j^{th} step. For simplicity, we suppress those arguments which remain constant during this inversion, letting $g_j(K_j) = g^{(j-1)}(\mathbf{z}_{j-1},\mathbf{K}_j)$ and $G_j(z_j) = g^{(j)}(\mathbf{z}_j,\mathbf{K}_{j+1})$. With this notation, the inversion formula (3.4) is

$$g_j(K_j) = \frac{1}{2 l_j K_j r_j^{K_j}} \sum_{k=-l_j K_j}^{l_j K_j - 1} G_j(r_j e^{\pi i k / l_j K_j}) e^{-\pi i k / l_j} - e_j\ , \tag{3.5}$$

where $i = \sqrt{-1}$, l_j is a positive integer r_j is a positive real number and e_j represents the *aliasing error*, which is given by

$$e_j = \sum_{n=1}^{\infty} g_j(K_j + 2 n l_j K_j) r_j^{2 n l_j K_j}\ . \tag{3.6}$$

Note that (3.5) differs from (2.3) of Choudhury, Leung and Whitt (1993), because that inversion formula was chosen to exploit Euler summation (involving nearly alternating series), whereas here we use simple truncation instead of Euler summation, as we explain in Section 5.1.

Note that, for $j = 1$, $g_1(K_1) = g(\mathbf{K})$ is real, so that $G_1(\overline{z}_1) = \overline{G_1(z_1)}$. This enables us to cut the computation in (3.5) by about one half; see Choudhury, Leung and Whitt (1993).

To control the aliasing error in (3.6), we choose $r_j = 10^{-a_j}$ for $a_j = \gamma_j / (2 l_j K_j)$. Then (3.6) becomes

$$e_j = \sum_{n=1}^{\infty} g_j(K_j + 2 n l_j K_j) 10^{-\gamma_j n}\ . \tag{3.7}$$

As is clear from (3.5), a bigger γ_j decreases the aliasing error. Also, as explained in Choudhury, Lucantoni and Whitt (1994a), the parameter l_j controls roundoff error, with bigger values causing less roundoff error. An inner sum of the inversion requires more accuracy than an outer sum since the inverted values in an inner sum are used as transform values in an outer sum.

With a goal of about eight significant digit accuracy, the following sets of l_j and γ_j typically are adequate: i) $l_1 = 1, \gamma_1 = 11$, ii) $l_2 = l_3 = 2$, $\gamma_2 = \gamma_3 = 13$, iii) $l_4 = l_5 = l_6 = 3$, $\gamma_4 = \gamma_5 = \gamma_6 = 15$, assuming that computations are done using double-precision arithmetic. It is usually not a good idea to use the same l_j for all j, because then more computation is done to achieve the same accuracy.

When the inverse function is a probability, the aliasing error e_j in (3.7) can easily be bounded. In contrast, here the normalization constants may be arbitrarily large and therefore the aliasing error e_j may also be arbitrarily large. Thus, we scale the generating function in each step by defining a *scaled generating function* as

$$\overline{G}_j(z_j) = \alpha_{0j} G_j(\alpha_j z_j) \ , \tag{3.8}$$

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 errors are suitably controlled. The inversion of $\overline{G}_j(z_j)$ in (3.8) yields the *scaled normalization constant*

$$\overline{g}_j(K_j) = \alpha_{0j} \alpha_j^{K_j} g_j(K_j) \ . \tag{3.9}$$

4. Scaling

The numerical inversion algorithm is completed by choosing the scaling parameters in (3.8). A general scaling strategy is described in Section 2.2 of Choudhury, Leung and Whitt (1993) and a detailed scaling algorithm for a class of closed queueing networks is developed in Section 5 there. Here we briefly describe the general scaling strategy in Section 4.1. In Section 4.2 we derive an appropriate scaling for a single resource with a single request class arriving according to a Poisson process. Then we develop a heuristic extension to multiple classes in Section 4.3. We use similar heuristics for multiple resources and non-CS sharing policies.

Since the scaling involves heuristics, we validate its effectiveness in two ways: First, we compare against alternative algorithms whenever available. Second, we perform calculations with different inversion parameters (corresponding to different contours in the inversion integral). Since the computations are indeed different, the accuracy is indicated by the extent of the agreement. Several examples appear in Sections 7-10. They show that the accuracy is always very high (8 or more decimal places).

4.1 General Scaling Strategy

We choose the parameters α_{0j} and α_j in (3.8) to control the *aliasing error with scaling*

$$\overline{e}_j = \sum_{n=1}^{\infty} \overline{g}_j(K_j + 2nl_j K_j) 10^{-\gamma_j n} \ . \tag{4.1}$$

Since we are interested in ratios of normalization constants, we focus on *relative errors* $e'_j = \overline{e}_j / \overline{g}_j(K_j)$, which can be bounded by

$$|e'_j| \le \sum_{n=1}^{\infty} \left| \frac{\overline{g}_j(K_j + 2nl_j K_j)}{\overline{g}_j(K_j)} \right| 10^{-\gamma_j n} \ . \tag{4.2}$$

Let

$$C_j \;=\; \max_n \left\{ \left| \frac{\overline{g}_j(K_j \;+\; 2nl_j K_j)}{\overline{g}_j(K_j)} \right| \right\}^{1/n} \;=\; \alpha_j^{2l_j K_j} \max_n \left\{ \left| \frac{g_j(K_j + 2nl_j K_j)}{g_j(K_j)} \right| \right\}^{1/n} \quad . \tag{4.3}$$

Then

$$| e_j' | \;\le\; \sum_{n=1}^{\infty} C_j^n 10^{-\gamma_j n} \;\le\; \frac{C_j 10^{-\gamma_j}}{1 - C_j 10^{-\gamma_j}} \;\approx\; C_j 10^{-\gamma_j} \;\; . \tag{4.4}$$

Note that C_j in (4.3) is independent of α_{0j}. We use the second parameter α_{0j} mainly to keep $\overline{g}_j(K_j)$ close to 1, so as to avoid numerical underflow or overflow. (This numerical problem also can be avoided by working with logarithms.)

Hence, our main goal is to choose α_j so that $C_j \;<<\; 10^{\gamma_j}$. Of course, in general we do not know $g_j(K_j)$ and thus we do not know C_j. However, we aim to achieve $C_j \;<<\; 10^{\gamma_j}$ by roughly controlling the growth rate of $\overline{g}_j(K_j)$, or its fastest growing term, exploiting the structure of the generating function.

4.2 Scaling for a Single Resource

In this section we do a careful analysis of the simple case of a single resource with a single class using the CS policy. By (2.12) and (3.8), the scaled generating function is

$$\overline{G}(z) \;=\; \alpha_0 \, \exp\, (\rho \alpha^a z^a)/(1 - \alpha z) \;, \tag{4.5}$$

where we have dropped subscripts. Through explicit inversion,

$$\overline{g}(K) \;=\; \alpha_0 \alpha^K \sum_{k=0}^{\lfloor K/a \rfloor} \rho^k / k! \;=\; \alpha_0 \alpha^K e^{\rho} F_{\rho}(\lfloor K/a \rfloor) \;, \tag{4.6}$$

where $\lfloor x \rfloor$ is the greatest integer less than or equal to x and $F_\rho(x) \;\equiv\; P(X \le x)$ is the cumulative distribution function (cdf) of a Poisson random variable X with mean ρ (which is unimodal with mode at an integer next to ρ).

Since $0.5 \le F_\rho(x) \le 1$ for $x > \rho$, if $\rho < K/a$, then $\alpha_0 = e^{-\rho}$ and $\alpha = 1$ yields $0.5 \le \overline{g}(l) \;\le\; 1$ for all $l \ge K$, so that $C \le 2$ in (4.3).

Next we treat the case $\rho \;>\; K/a$. Here our strategy is to control the growth rate of the largest term of $\overline{g}(K)$ with respect to K (which happens to be the last term). Using Stirling's formula for the factorial, the *largest term* of $\overline{g}(K)$ is

$$\alpha_0 \alpha^K e^{\rho} \left[\frac{e^{-\rho} \rho^{\lfloor K/a \rfloor}}{\lfloor K/a \rfloor !} \right] \;\approx\; \frac{\alpha_0 (\alpha^a \rho)^{K/a}}{(K/a)^{K/a} e^{-K/a} \sqrt{2\pi K/a}} \;\; . \tag{4.7}$$

We use the term involving α in the numerator to cancel the dominant term $(K/a)^{K/a}$ in the denominator and α_0 to cancel the next dominant term $\exp(-K/a)$ in the denominator. This yields the following scale factors:

$$\alpha = (K/a\rho)^{1/a} \quad \text{and} \quad \alpha_0 = e^{-K/a} = e^{-\rho\alpha^a} . \tag{4.8}$$

Alternatively, we could use α alone to cancel out the two most dominant terms in the denominator of (4.7) and use α_0 to cancel the remaining term in the denominator to get the scaling

$$\alpha = (K/a\rho e)^{1/a} \quad \text{and} \quad \alpha_0 = \sqrt{2\pi K a} . \tag{4.9}$$

We choose (4.8) instead of (4.9) to ensure that the scale parameters change smoothly with K and coincide with the scale parameters in the other case at $\rho = K/a$. The scaling in the two cases may be combined as follows:

$$\alpha \text{ is the largest number in } (0,1] \text{ such that } \rho\alpha^a a \le K \quad \text{and} \quad \alpha_0 = \exp(-\rho\alpha^a) . \tag{4.10}$$

4.3 Heuristic Extensions

With multiple classes and a single resource, we extend (4.10) heuristically by stipulating that

$$\alpha \text{ is the largest number in } (0,1] \text{ such that } \sum_{j=1}^{r} \rho_j \alpha^{a_j} a_j \le K \tag{4.11}$$

and letting

$$\alpha_0 = \exp\left(-\sum_{j=1}^{r} \rho_j \alpha^{a_j}\right) . \tag{4.12}$$

In order to get α satisfying (4.11), we first compute $\sum_{j=1}^{r} \rho_j a_j$. If this quantity is less than or equal to K, then we immediately get $\alpha = 1$. Otherwise α is obtained as the unique solution in the interval $(0,1)$ of the non-linear equation $\sum_{j=1}^{r} \rho_j \alpha^{a_j} a_j = K$. Any search procedure, such as bisection, is adequate since we do not need to find α with high precision.

It is to be noted that the heuristic extension satisfies two consistency checks. First, for $r = 1$, (4.11) and (4.12) reduce to (4.10). Second, if a_j is independent of j, then (4.11) and (4.12) again coincide with (4.10) with $\rho = \sum_{j=1}^{r} \rho_j$ and $a = a_j$. This is a nice check since in this case the independent Poisson streams may be combined to a single one for the blocking probability calculation.

Turning to multiple resources, from (2.12), (3.8) and (3.9), the scaled generating function is

$$\overline{G}(z_1, \ldots, z_p) = \frac{\prod_{i=1}^{p} \alpha_{0i} \exp\left\{ \sum_{j=1}^{r} \rho_j \prod_{i=1}^{p} (\alpha_i z_i)^{a_{ij}} \right\}}{\prod_{i=1}^{p} (1 - \alpha_i z_i)} \tag{4.13}$$

and the scaled normalization constant is

$$\bar{g}(\mathbf{K}) \;=\; \prod_{i=1}^{p} (\alpha_{0i} \alpha_i^{K_i}) \, g(\mathbf{K}) \;. \tag{4.14}$$

Using similar heuristics as above, we obtain the scaling parameters α_i, $1 \le i \le p$, so that they satisfy the inequalities $0 < \alpha_i \le 1$ and

$$\sum_{j=1}^{r} \rho_j \prod_{k=1}^{p} \alpha_k^{a_{kj}} \prod_{k=1}^{i-1} r_k^{a_{kj}} a_{ij} \le K_i \;, \quad 1 \le i \le p \;, \tag{4.15}$$

where r_k is as defined in Section 3.2. Once the scaling variables α_i have been obtained, we obtain α_{0i} recursively starting with $i = p$ by

$$\prod_{k=i}^{p} \alpha_{0k} \;=\; \exp \left\{ - \sum_{j=1}^{r} \rho_j \prod_{k=1}^{p} \alpha_k^{a_{kj}} \prod_{k=1}^{i-1} r_k^{a_{kj}} \right\} \;. \tag{4.16}$$

To find a maximal vector $(\alpha_1 , \ldots , \alpha_p)$ satisfying (4.15), we start with $i = p$ and successively decrease *i*. We use the fact that the left side of (4.15) is monotone in α_i. When $i = l$, the values of α_i for $i \ge l + 1$ are known. We approximate by acting as if $\alpha_i = 1$ for $i \le l - 1$ and find α_l satisfying the constraint for $i = l$ in (4.15). When we are done we obtain a maximal vector; i.e., if any α_i is less than 1, then at least one constraint in (4.15) is necessarily satisfied as an equality. Hence, we cannot increase the vector without violating a constraint. However, in general there may be many maximal vectors.

For the UL policy the generating function in (2.17) has a form identical to the CS generating function in (2.12) if we treat the y_j variables like the z_i variables. Thus the scaling for UL is straightforward extension of CS scaling.

The generating function for the GM policy in (2.20) is more complicated. Hence, for scaling only, we treat GM by acting as if it were UL. For each class *j* on each resource *i*, we use the upper limit obtained by subtracting the guaranteed minima of all other classes from the capacity. We then use the UL scaling with these upper limit parameters. This procedure is exact for two classes, but a heuristic approximation for more than two classes.

5. Other Algorithm Issues

In this section we consider other algorithm issues besides scaling. In Section 5.1 we discuss truncation; in Section 5.2 we discuss speeding up the algorithm when there are many classes by exploiting shared computation; and in Section 5.3 we discuss computational complexity.

5.1 Truncation

As can be seen from (3.5), the inversion formula in each dimension is a sum of $2 l_i K_i$ terms. If K_i is large, then it is natural to look for ways to accelerate convergence of the finite sum. For the closed queueing network models in Choudhury, Leung and Whitt (1993), we found that we could usually exploit Euler summation for this purpose. Here we find that Euler summation is usually not effective, but that truncation is. It is possible that other acceleration techniques will be even more effective, but we leave that to future work. (See Wimp (1981) for candidates.)

The inversion formula in each dimension is a weighted sum of generating function values evaluated over equidistant points along the circumference of a circle. The weights are complex numbers, but they have constant amplitude. As the capacities K_i grow, the amplitude of the generating function typically becomes unevenly distributed along the circumference of the circle. There are several local maximum points and the amplitude drops sharply away from these points. (Since the weights have constant amplitude, we only need to consider the relative amplitude of the generating function values.) If we can identify all the relative maximum points, and then consider only those points around them that have non-negligible relative amplitude, we can obtain a significant reduction in computation.

We first develop the truncation procedure for a single resource and then consider the extension to multiple resources. We start with the scaled generating function in (4.5), based on (2.12) and (3.8). As noted after (3.6), in the outer dimension we can cut the computation in half; we thus consider the sum over the upper semicircle with radius $r_1 = 10^{-\gamma_1/2l_1K_1}$.

At a summation point, $z_1 = r_1 e^{i\theta}$ where θ assumes the values $\pi k/(l_1 K_1)$ for $0 \le k \le l_1 K_1$. Let $\overline{G}^*(\theta)$ be $\overline{G}(z_1)$ expressed as a function of θ, i.e.,

$$\overline{G}^*(\theta) = \frac{\alpha_{01} \prod_{j=1}^{r} \exp(\rho_j \alpha_1^{a_j} r_1^{a_j} e^{i a_j \theta})}{1 - \alpha_1 r_1 e^{i\theta}} \ . \tag{5.1}$$

Note that the amplitude of $\overline{G}^*(\theta)$ in (5.1) is

$$|\overline{G}^*(\theta)| = \frac{\alpha_{01} \prod_{j=1}^{r} \exp(\rho_j (\alpha_1 r_1)^{a_j} \cos(a_j \theta))}{\sqrt{1 + \alpha_1^2 r_1^2 - 2\alpha_1 r_1 \cos\theta}} \ . \tag{5.2}$$

For $j = 1,2,\ldots,r$, the numerator has relative maxima at $\theta = 2 l_j \pi / a_j$ for $l_j = 0,1,\ldots,\lfloor a_j/2 \rfloor$. The denominator has a single minimum at $\theta = 0$. Hence, $|\overline{G}^*(\theta)|$ has a *global maximum* at $\theta = 0$ and *potential local maxima* at $\theta = 2 l_j \pi / a_j$ for $l_j = 1,2,\ldots,\lfloor a_j/2 \rfloor$ and $j = 1,2,\ldots,r$. Note that usually many of these $\sum_{j=1}^{r} \lfloor a_j/2 \rfloor$ local maxima will coincide.

In summary, we start by computing $|\overline{G}^*(0)|$ and then find the distinct local maximum points $\theta = 2 l_j \pi / a_j$ for $l_j = 1,\ldots,\lfloor a_j/2 \rfloor$ and $j = 1,\ldots,r$, and sort them in increasing order. Let these points be θ_m^i for $i = 1,2,\ldots,L$. In general θ_m^i may not coincide with a summation point in the inversion algorithm. In that case, move θ_m^i to the nearest summation point used in the inversion algorithm. Next find all i such that $|\overline{G}^*(\theta_m^i)/\overline{G}^*(0)| \ge \varepsilon$, where ε is some allowable error. Then

$$|\overline{G}^*(\theta)/\overline{G}^*(0)| = \exp\left(-\sum_{j=1}^{r} \rho_j (\alpha_1 r_1)^{a_j} (1 - \cos a_j \theta)\right) \frac{1 - \alpha_1 r_1}{\sqrt{1 + \alpha_1^2 r_1^2 - 2\alpha_1 r_1 \cos\theta}} \ . \tag{5.3}$$

For all these i sum over all summation points in the inversion algorithm above and below θ_m^i until $|\overline{G}^*(\theta)/\overline{G}^*(0)| \ge \varepsilon$. Do not sum over any summation point more than once.

Our experience is that for large ρ_j and K_1 typically only the points around $\theta = 0$ and a few other local maxima will be significant. We can see how much computational savings will result by computing for all values of θ in $(0,\pi)$ and finding the proportion of the range for which

$|\overline{G}^*(\theta)/\overline{G}^*(0)| > \varepsilon$. To illustrate, we do an example.

Example 5.1. Let $r = 2$, $a_1 = 1$, $a_2 = 2$, $\rho_1 = K_1/2$, $\rho_2 = K_1/4$ and let K_1 be variable. In Table 1 below we show the proportion of the range $[0,\pi]$ for which $|\overline{G}^*(\theta)/\overline{G}^*(0)| \geq \varepsilon$ for various values of K_1 and ε. Clearly, the smaller the proportion, the bigger the savings. In fact, the computational saving is the inverse of the proportion.

K_1	$\varepsilon = 10^{-6}$	$\varepsilon = 10^{-8}$	$\varepsilon = 10^{-10}$	$\varepsilon = 10^{-12}$
20	1.000	1.00	1.00	1.00
50	0.24	0.29	0.34	0.39
100	0.15	0.17	0.20	0.22
1000	0.040	0.047	0.053	0.059
10,000	0.012	0.014	0.016	0.018
100,000	0.0035	0.0043	0.0050	0.0055

Table 1. Proportion of θ values for which $|\overline{G}^*(\theta)/\overline{G}^*(0)| \geq \varepsilon$ as a function of ε and K_1 for the two-class example.

From Table 1, we see that the savings increases rapidly as K increases, and decreases slowly as ε decreases. There is no savings at $K_1 = 20$, but a savings by a factor of 182 even with a low error tolerance of $\varepsilon = 10^{-12}$ at $K_1 = 10^5$. From Table 1, it appears that the savings is approximately proportional to $\sqrt{K_1}$. Equivalently, this means that with truncation the required summation is approximately proportional to $\sqrt{K_1}$ instead of K_1.

To see that the required summation should be $O(\sqrt{K_1})$ more generally, focus on the exponential term in (5.3) and note that it is approximately of the form

$$\exp(C_1 K_1 (1-\cos(k/C_2 K_1))) \approx \exp(C_1 k^2 / 2C_2^2 K_1)$$

for $\theta = \pi k/l_1 K_1$ and C_1 and C_2 constants (using $1-\cos x \approx x^2/2$ for x small). Thus, the number of θ values satisfying $|\overline{G}^*(\theta)/G^*(0)| > \varepsilon$ should be $O(\sqrt{K_1})$ as K_1 gets large.

Truncation can also be exploited with multiple resources, but the situation is more complicated. Now the scaled generating function is given by

$$\overline{G}(z_1, \ldots, z_p) = \frac{\prod_{i=1}^{p} \alpha_{0i} \exp\left[\sum_{j=1}^{r} \rho_j \prod_{i=1}^{p} (\alpha_i z_i)^{a_{ij}}\right]}{\prod_{i=1}^{p} (1-\alpha_i z_i)} \tag{5.4}$$

and the scaled normalization constant is

$$\overline{g}(\mathbf{K}) = \prod_{i=1}^{p} (\alpha_{0i} \alpha_i^{K_i}) g(\mathbf{K}) \ . \tag{5.5}$$

For inversion with respect to z_p, the same computational saving as for a single resource may be achieved, but there are two differences: First, the inversion formula involves summation over the entire circle instead of just a semicircle, so that we have to consider more maximum points. Second, the constant $\rho_j \alpha^{a_j}$ in (4.12) has to be replaced by the constant $\rho_j \prod_{l=1}^{p} \alpha_l^{a_{lj}} \prod_{l=1}^{p-1} z_l^{a_{lj}}$. Since the latter constant is a complex number, it will introduce a constant phase change to θ and hence to all maximum points.

For inversion with respect to z_i for $i < p$, we have to cope with the partially inverted generating function $g^{(j-1)}(\mathbf{z}_{j-1}, \mathbf{K}_j)$ in (3.4), for which we do not know the functional form. Hence, we do not know the maximum points and we must resort to heuristics. We have observed that by heuristically assuming the location of the maximum points to be the same as if the partially inverted generating function has the same functional form as in (5.4) usually works and gives good computational savings. Obviously, care should be taken with these truncations, because we no longer have full control of the errors. This is a good direction for further research. Perhaps asymptotic analysis can be used to develop more effective truncation.

5.2 Efficient Computation For Many Classes

In order to compute the blocking probability for each of the r classes, the computational complexity is $O(r^2)$, because $r + 1$ normalization constant values have to be calculated and the computation required for each is $O(r)$. However, we will show that for large capacity vectors $\mathbf{K}$ it is possible to compute the $r + 1$ normalization constants simultaneously with the bulk of the computations shared, so that the required computation for all normalization constants is only slightly more than for one. This reduces the overall computational complexity from $O(r^2)$ to $O(r)$.

To explain the method, we consider a single resource and the CS policy. From (2.11), we know that the blocking probability for class j is $B_j = 1 - g(K_1 - a_j)/g(K_1)$. Then, letting $a_0 \equiv 0$, we must compute $g(K_1 - a_j)$ for $0 \le j \le r$. Combining the scaling and inversion procedure in Sections 3.2 and 4, we see that the standard formula for this computation is

$$g(K_1 - a_j) = \frac{\alpha_{01j}^{-1} \alpha_{1j}^{-(K_1 - a_j)}}{m_{1j} r_{1j}^{K_1 - a_j}} \sum_{k=-\frac{m_{1j}}{2}}^{\frac{m_{1j}}{2}-1} \alpha_{01j} G(\alpha_{1j} r_{1j} e^{\frac{2\pi ik}{m_{1j}}}) e^{-\frac{2\pi ik(K_1 - a_j)}{m_{1j}}} , \tag{5.6}$$

where α_{01j} and α_{1j} are the scaling parameters (which may be obtained from (4.11) and (4.12)),

$$m_{1j} = 2l_1(K_1 - a_j) \quad \text{and} \quad r_{1j} = 10^{-\gamma_1/m_{1j}} . \tag{5.7}$$

The associated aliasing errors are

$$e_{1j} = \sum_{n=1}^{\infty} \alpha_{01j} \alpha_{1j}^{K_1 - a_j} g(K_1 - a_j + nm_{1j}) 10^{-n\gamma_1} . \tag{5.8}$$

Note that the computation (5.6) for different values of j cannot be shared because the quantities α_{01j}, α_{1j} and m_{1j} are different for different values of j.

In order to share computation for different values of j, we propose that for $K_1 >> a_j$ $(0 \le j \le r)$, the quantities α_{01j}, α_{1j} and m_{1j} be replaced by their values at $j = 0$ for all j. This should not cause any appreciable difference in the error expression (5.8) since for $K_1 >> a_j$ the quantities α_{01j}, α_{1j} and m_{1j} are pretty close to their values at $j = 0$ anyway.

Dropping the subscript j for α_{01j}, α_{1j} and m_{1j}, we can rewrite (5.6) as

$$g(K_1 - a_j) = \frac{1}{m_1 \alpha_{01} (\alpha_1 r_1)^{K_1 - a_j}} \sum_{k=-\frac{m_1}{2}}^{\frac{m_1}{2}} T_k C_j^k , \tag{5.9}$$

where

$$C_j = e^{\frac{2\pi i a_j}{m_1}} \quad \text{and} \quad T_k = \alpha_{01} G(\alpha_1 r_1 e^{\frac{2\pi i k}{m_1}}) e^{-\frac{2\pi i k K_1}{m_1}} . \tag{5.10}$$

Note that the bulk of the computation in (5.9) is computing T_k which can be shared. The quantities C_j^k may be computed quickly, since C_j needs to be computed only once for each j. It is also clear that by working with partial sums for all j simultaneously the overall computation may be done with a storage requirement $O(r)$. Moreover, if truncation applies (see Section 5.1), then it applies uniformly for all j since $|C_j^k| = 1$. For multiple resources and other sharing policies, the same approach works.

5.3 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 or GM. The main computational burden is carrying out the p-fold nested inversion in (3.5). 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 $r + 1$ normalization constants, but in Section 5.2 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 we have shown that we can use truncation to reduce K to $\bar{K} << K$ and, with special structure (see Section 3.1), we can reduce p to $\bar{p} << p$. So, finally, we get

$$C_{CS} = O(r\bar{K}^{\bar{p}}) , \tag{5.11}$$

where

$$\begin{aligned} &\bar{K} \le K \text{ and } \bar{K} << K \text{ for large } K \\ &\bar{p} \le p \text{ and } \bar{p} << p \text{ with special structure} . \end{aligned} \tag{5.12}$$

By contrast, the computational complexity of the algorithms in Dziong and Roberts (1987) (in the Poisson case) and Pinsky and Conway (1992) are

$$C_{CS} = O(rK^p) . \tag{5.13}$$

So our algorithm would be faster if we can exploit special structure or if K is large, where we can exploit truncation. Conway and Pinsky (1992) also reduces p by using special structure (although differently from ours) but they do not seem to reduce K.

For the upper limit and guaranteed minimum sharing policies (exploiting conditional decomposition as described in Section 3.1), we get

$$C_{UL} = C_{GM} = O(r\bar{K}^{\bar{p}+1}) \tag{5.14}$$

for $\bar{K}$ and $\bar{p}$ in (5.12). For these policies there does not appear to be a general recursive algorithm for multiple resources. For a single resource, Chuah (1993) has developed a recursion for the UL policy with complexity $O(r^2 K^2)$; this is slower than ours for large K.

The storage requirement for our inversion algorithm is $O(r)$ for all cases. This is an attractive feature of our algorithm since it is much smaller than the storage requirement of the recursive algorithms.

6. An Alternative Direct Algorithm

In order to validate our inversion algorithm on small models, we also constructed a direct algorithm. We directly calculate the normalization constant $g(\mathbf{K})$ via the r-fold nested sum

$$g(\mathbf{K}) = \sum_{n_1=0}^{n_{1u}} \frac{\rho_1^{n_1}}{n_1!} \sum_{n_2=0}^{n_{2u}} \frac{\rho_2^{n_2}}{n_2!} \cdots \sum_{n_r=0}^{n_{ru}} \frac{\rho_r^{n_r}}{n_r!} , \tag{6.1}$$

implemented as an r-level nested do loop. The upper limits of summation n_{ju} depend on the sharing policy and also on n_k for $1 \le k \le j-1$. (This is why the sums cannot be separated, but must be treated in a nested fashion.) In each case, when considering the j^{th} sum, the values of n_k for $1 \le k \le j-1$ are already fixed. The maximum value n_j can take occurs when $n_k = 0$ for $k \ge j + 1$. Therefore we easily obtain the proper upper limits n_{ju}. For CS,

$$n_{ju} = \min_{\substack{1 \le i \le p \\ a_{ij} \ne 0}} \left\{ \left\lfloor (K_i - \sum_{k=1}^{j-1} n_k a_{ik})/a_{ij} \right\rfloor \right\} . \tag{6.2}$$

For UL, we apply (2.14) to obtain

$$n_{ju} = \min\{M_j, \min_{\substack{1 \le i \le p \\ a_{ij} \ne 0}} \left\{ \left\lfloor (K_i - \sum_{k=1}^{j-1} n_k a_{ik})/a_{ij} \} \right\rfloor \right\} . \tag{6.3}$$

For GM, we apply (2.18) to obtain

$$n_{ju} = \min_{\substack{1 \le i \le p \\ a_{ij} \ne 0}} \left\{ \left\lfloor (K_i - \sum_{k=1}^{j-1} \max\{n_k a_{ik}, \delta_{ik} M_k\} - \sum_{k=j+1}^{r} \delta_{ik} M_k)/a_{ij} \right\rfloor \right\} . \tag{6.4}$$

Let $x_{jk} = \rho_j^k / k!$ The quantities x_{jk} can be calculated recursively by setting $x_{j0} = 1$ and

$$x_{jk} = x_{j,k-1} (\rho_j / k) \ . \tag{6.5}$$

To speed up the computation, all the x_{jk}'s are precomputed, stored and used as needed for $0 \le k \le n_{ju}^{\max}$, where $n_{ju}^{\max}$ is the maximum possible value of n_{ju}, which is obtained from (6.2)–(6.4) by setting $n_k = 0$ for $1 \le k \le j-1$. The total storage required for the CS policy is $\sum_{j=1}^{r} \left[\min_{1 \le i \le p} \left[\lfloor K_i / a_{ij} \rfloor \right] \right]$ and less for the other policies. Assuming for simplicity that $K_i = K$ for all i and $a_{ij} = 1$, and keeping in mind that $g(K)$ has to be computed for $(r+1)$ sets of parameter values, the overall complexity is $O(rK^r)$.

For large capacity values there may be numerical underflow/overflow problems. Instead of always starting the summation over n_j at 0 as in (6.1), it may be more efficient to start it near ρ_j since x_{jk} has a maximum near $k = \rho_j$ (and then use truncation when k is far from ρ_j). (Special care is needed, since in some cases n_{ju} may be much smaller than ρ_j.) We do not address these issues in this paper.

Looking at the complexity and storage requirements of the direct algorithm, we see that it will be effective only when r is small. However, with special model structure, many of the inner summations in (6.1) become independent of each other, thereby allowing efficient computation even when r is very large; i.e., once again we are able to apply conditional decomposition based on special structure to reduce the effective dimension of the computation.

One such structure is the dual of the two-hop tree network: There are p resources and $r = p+1$ classes of requests. Class 1 requires one or more units from each resource, while class j requires one or more units only from resource $j-1$ for $2 \le j \le p+1$. Formula (6.2) simplifies to

$$n_{ju} = \lfloor (K_{j-1} - n_1) / a_{j-1,j} \rfloor \ , \ 2 \le j \le p+1 \ . \tag{6.6}$$

If we fix n_1, then all the n_{ju}'s are fixed and independent of each other, so that the sums in (6.1) become independent of each other. Therefore, the overall problem requires only a two-level nested summation, so that it can be solved even when r, p and K are all large.

Example 6.1. We now give an example illustrating how the direct algorithm can perform with such special structure. (We give similar examples for the inversion algorithm in Section 10.) In our example $p = 10$, $r = 11$, $K_i = 45 + 5i$ and $a_{i1} = 1$, $1 \le i \le p$, $a_{j-1,j} = 1$ for $2 \le j \le 6$ and $a_{j-1,j} = 2$ for $7 \le j \le 11$, and the vector of offered loads is

$$\boldsymbol{\rho} = (25, 30, 35, 40, 45, 50, 27, 30, 33, 36, 39) \ .$$

We consider the CS and UL policies. For the UL policy, the upper limit vector is

$$\mathbf{M} = (50, 30, 35, 40, 45, 50, 27, 30, 33, 36, 39) \ .$$

The blocking probabilities for classes 1,2,10 and 11 are given in Table 2.

class	CS	UL
1	0.4056451	0.2744358
2	0.0579203	0.1363239
10	0.1023460	0.1327021
11	0.1062107	0.1334783

Table 2. Blocking probabilities computed by the direct algorithm, exploiting dimension reduction.

The computational complexity for computing all blocking probabilities is $O((r+1)K_1 \sum_{j=2}^{r} K_j)$, with the leading $r+1$ due to needing $r+1$ normalization constant values. Hence, it is $O(r^2 K^2)$. Computing all blocking probability for this example took only a fraction of a second. We could solve examples substantially bigger as well (e.g. larger capacities), but then it is necessary to avoid numerical underflow/overflow problems by working with logarithms or scaling (which we do not do here).

7. Examples of a Single Resource with Complete Sharing

We now give examples illustrating the numerical inversion algorithm. All computations were done on a SUN SPARC-2 workstation.

Our first numerical example is the classical resource-sharing model with a single resource and the CS sharing policy. The generating function is given in (2.12) with $p = 1$. This example is relatively elementary since the generating function is one-dimensional. As a basis for comparison, we also implemented the recursive algorithms of Kaufman (1981) and Roberts (1981) and the uniform asymptotic approximation (UAA) of Mitra and Morrison (1993).

The specific model we consider has 2 classes of requests with requirements $a_{1,1} = 1$ and $a_{1,2} = 12$. The capacity K ranges from 20 (relatively small) to 5,000,000 (very large). We consider three regions: heavily loaded, critically loaded and lightly loaded. These regions occur as the total offered load $\sum_{j=1}^{r} \rho_j a_{1j}$ is significantly greater than, approximately equal to and significantly less than the capacity K, respectively. In the heavily loaded region the offered loads are $\rho_1 = 0.15K$ and $\rho_2 = 0.10K$ for classes 1 and 2, so that $\sum_{j=1}^{2} \rho_j a_{1j} = 1.35K$ in all heavily loaded cases. For the critically loaded region, we let $\sum_{j=1}^{2} \rho_j a_{1j} \approx K$, but we avoid exact equality to avoid degeneracy in the UAA algorithm. (Mitra and Morrison (1993) propose an alternate formula for this degenerate case, but we do not use it.) For the lightly loaded region, we set $\Sigma \rho_j a_{1j} \approx K - 4.313\sqrt{K}$. (It is known that the total offered load should grow as $K - c\sqrt{K}$ for some constant c for the blocking to be non-negligible.) The specific parameter values appear in Table 3.

We give numerical results for the blocking probabilities of each class in Table 3. These are based on formula (2.11). In the cases with $K \le 500$, the inversion and recursion algorithms agreed well beyond the accuracy given (eight significant digits). For $K \ge 5000$, the recursion either had numerical underflows or overflows, or took too long to run. For $K \ge 5000$, the inversion results agreed closely with UAA, and, as expected, the agreement improves as K increases. However, in the heavily loaded region, for $K \geq 500{,}000$ even the UAA had numerical overflow problems. Also, in some cases of the critical region, the UAA was too close to its degenerate region. In each case we also checked the accuracy of the inversion algorithm directly by running it twice, with $l_1 = 1$ and $l_1 = 2$.

In all cases considered here the inversion algorithm took less than a second. For the larger values of K, a critical factor in achieving this speed is truncation. It is significant that the computational complexity in calculating all the blocking probabilities grows only linearly with the number of classes. So the inversion algorithm can solve models with very large number of classes (say, 100) and very large K (say, 5,000,000) in only tens of seconds.

For the example in Table 3 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 3 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 significantly increase $a_{1,2}$, e.g., from 12 to 100 or more. An example with finite sources showing the limitations of UAA is given in Choudhury, Leung and Whitt (1994a).

parameters			blocking probability of class 1			blocking probability of class 2		
K	ρ_1	ρ_2	inversion	recursion	UAA	inversion	recursion	UAA
heavily loaded region								
20	3.0	2.0	5.4147360e-3	same	6.38e-2	0.6670903	same	0.6580
50	7.5	5.0	1.6157470e-2	same	4.92e-2	0.5259738	same	0.5080
500	75	50	3.2084709e-2	same	3.2151e-2	0.3317448	same	0.3317368
5,000	750	500	2.8456587e-2	—	2.84627e-2	0.2936775	—	0.2936770
50,000	7,500	5,000	2.7984241e-2	—	2.7984851e-2	0.2887462	—	0.2887461
500,000	75,000	50,000	2.7935032e-2	—	—	0.2882328	—	—
5,000,000	750,000	500,000	2.7930089e-2	—	—	0.2881812	—	—
critically loaded region								
50	7.5	3.5	9.9887360e-3	same	3.18e-2	0.39930413	same	0.383
500	75	35	1.0194371e-2	same	1.0217e-2	0.1232678	same	0.1232615
5000	750	350	2.8928545e-3	–	2.89351e-3	3.4908258e-2	–	3.490795e-2
50,000	7,500	3,530	9.404766e-4	—	9.40488e-4	1.130281e-2	—	1.130269e-2
500,000	75,000	35,400	3.262452e-4	—	—	3.915783e-3	—	—
5,000,000	750,000	354,100	1.012373e-4	—	—	1.214952e-3	—	—
lightly loaded region								
50	7.5	1	9.4989161e-4	same	2.71e-3	6.2853438e-2	same	6.14e-2
500	155.2	20.7	1.6906537e-3	same	1.6949e-3	2.3605271e-2	same	2.36040e-2
5000	1805.8	240.8	6.1636276e-4	–	6.1652e-4	7.7329048e-3	–	7.73287e-3
50,000	18,860	2,515	2.046621e-4	—	2.046674e-4	2.489884e-3	—	2.4898832e-3
500,000	191,137	25,488	6.713379e-5	—	6.713395e-5	8.090529e-4	—	8.090528e-4
5,000,000	1,919,390	255,947	2.284043e-5	—	2.284040e-5	2.744436e-4	—	2.744440e-4

Table 3. Numerical results for a single resource with two classes and the CS policy.

8. Examples with Different Sharing Policies

In this section we again consider a single resource, but now we consider the UL and GM sharing policies as well as the CS policy.

The generating functions for the UL and GM policies are given in (2.17) and (2.20) with $p = 1$, and the blocking probabilities are given in (2.15) and (2.19). Since $p = 1$, we can use either one-dimensional or two-dimensional inversion for the UL and GM policies, as explained in Section 3.1. For all examples in this section we use the one-dimensional inversion, exploiting the explicit inversions in (3.1) and (3.2).

We first consider a smaller example, for which we can apply the direct algorithm in Section 6 as well as the inversion algorithm. We let the capacity be $K = 150$ and consider 5 classes. The 5 classes require $1,2,3,4$ and 5 resource units per request, respectively, and the corresponding offered loads are $20,15,12,10$ and 9. For UL, the class limits are $20,30,50,60$ and 70. For GM, the corresponding guaranteed minima are $5,18,25,36$ and 40.

The blocking probability for each class and each sharing policy is shown in Table 4. The two computational methods agreed to at least 12 digits in each case. *This example thus serves to validate the generating functions and both algorithms.* For this smaller example, the inversion algorithm runs in less than a second while the direct algorithm runs in minutes. Hence, the direct algorithm is already beginning to reach its limits with this example.

sharing policy	class				
	1	2	3	4	5
CS	0.0605131	0.1188490	0.1749724	0.2288574	0.2804872
UL	0.1760449	0.2145673	0.1621870	0.1957472	0.2391045
GM	0.1482263	0.2542774	0.2851976	0.2167977	0.2441586

Table 4. Blocking probabilities for the three sharing policies and $K = 150$.

Next we consider a larger example with capacity $K = 600$ and 10 classes, for which the inversion algorithm runs in seconds, while the direct algorithm can no longer run in reasonable time. Class j requires j units, $1 \leq j \leq 10$. The vector of offered loads for the 10 classes is $(30,25,20,18,16,14,13,12,11,10)$. The vector class limits with the UL policy is $(30,50,60,80,90,100,110,120,130,140)$, while the associated vector of guaranteed minima for the GM policy is $(5,10,20,30,40, 50,60,70,80,100)$.

The blocking probabilities for classes 1, 2 and 10 for each sharing policy are given in Table 5. For the CS policy, the inversion algorithm was validated by applying the Kaufman (1981) and Roberts (1981) recursion. For the UL and GM policies, the inversion was validated by performing two separate inversions with the parameters $l_1 = 1$ and $l_1 = 2$. There was agreement to at least 11 digits in each case.

sharing policy	class 1	2	10
CS	0.0427417	0.0838918	0.3613882
UL	0.1455476	0.1706272	0.3199203
GM	0.0973615	0.1861317	0.1865667

Table 5. Blocking probabilities for the three sharing policies and $K = 600$.

9. The Kelly Example

We now consider the Kelly example in Figure 1 (in Section 1) with 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 unit requirements. We keep the CS policy and Poisson arrivals, but consider the multi-rate generalization. Furthermore, we allow multiple classes with different multi-rate requirements.

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 $1 \equiv r_0 < r_1 < r_2 \ldots < r_6 \equiv r$. 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 is

$$G(z) = \frac{1}{\prod_{i=1}^{6}(1-z_i)} \exp\Bigg[\sum_{j=1}^{r_1}\rho_j z_1^{a_{ij}} + \sum_{j=r_1+1}^{r_2}\rho_j z_2^{a_{2j}} + \sum_{j=r_2+1}^{r_3}\rho_j z_1^{a_{1j}} z_2^{a_{2j}} + \sum_{j=r_3+1}^{r_4}\rho_j z_3^{a_{3j}} z_5^{a_{5j}}$$

$$+ \sum_{j=r_4+1}^{r_5}\rho_j z_4^{a_{4j}} z_5^{a_{5j}} + \sum_{j=r_5+1}^{r}\rho_j z_1^{a_{1j}} z_3^{a_{3j}} z_5^{a_{5j}}\Bigg] \qquad (9.1)$$

The possible dimension reduction for this example is somewhat less obvious, so that it is helpful to use the systematic procedure in Section 3.1. That analysis shows that the dimension can be reduced from 5 to 3 by designating z_1 and z_5 as 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. Since the final dimension is 3, this example requires more computation than the previous two examples. For the optimal inversion order, we use the l_j parameter vectors $(1,2,3,3,3)$ and $(1,3,3,3,3)$ in the inversion.

The specific example we consider has 5 resources, as in Figure 1, with capacities $K_i = 15$ for all i. There are 12 classes, with two classes using each of the 6 routes. The specific offered loads are given in Table 6. We let the requirements be either 1 or 2 for each request, with each request having the same requirements on each resource (trunk). The specific requirements are also given in Table 6.

The blocking probabilities for each class in these three cases are given in Table 6. The computation of the blocking probabilities in Table 6 took about 20 seconds. 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.

10. Examples of Networks with Special Structure

Many loss models have special structure allowing drastic dimension reduction. We show two such structures in Figure 2 below, but clearly others are also possible. Figure 2 shows the resources.

In structure A, commonly referred to as a tree network, class j requires a_{ij} units from resource i, where a_{ij} is allowed to be non-zero only for at most *two* values of i, one of which has to be p. If both the non-zero values of a_{ij} are 1 then that corresponds to a single-rate tree network, considered by Mitra (1987) and Kogan (1989). Tsang and Ross (1990) considered the multi-rate case in which the two non-zero values of a_{ij} are the same but may be bigger than 1. Ross and Tsang (1990) allow the two possible non-zero values of a_{ij} to be either the same or one of them to be zero. We consider a further generalization by allowing the two values of a_{ij} to be different, without necessarily requiring one of them to be zero. Furthermore, all the earlier work was restricted to the CS policy, whereas we allow the UL and GM sharing policies.

class parameters				blocking probabilities for each class
j	ρ_j	route	rqmts.	Poisson arrivals
1	2	1	1	0.069930
2	1	1	2	0.155912
3	2	2	1	0.011306
4	1	2	2	0.030032
5	2	1,2	1	0.079276
6	1	1,2	2	0.176021
7	2	3,5	1	0.071961
8	1	3,5	2	0.159575
9	2	4,5	1	0.071961
10	1	4,5	2	0.15957
11	2	1,3,5	1	0.134236
12	1	1,3,5	2	0.280670

Table 6. Blocking probabilities in the Kelly example with the CS policy.

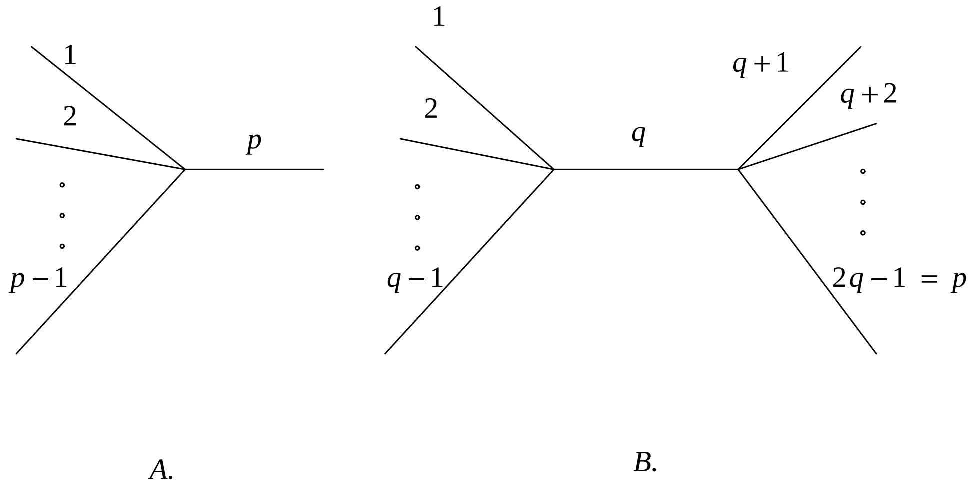

Figure 2. *Two Networks with Special Structure*

In structure B, evidently not considered earlier, a_{ij} is allowed to be non-zero for at most *three* values of *i*, one of which has to be *q* and the other two have to be *k* and $q + k$, for $1 \le k \le q-1$.

In structure A, divide the traffic classes according to the resources they use by letting $0 \equiv r_0 < r_1 < r_2 < \ldots < r_{p-1} \equiv r$. Then class *j* uses resources *k* and *p* if $r_{k-1}+1 \le j \le r_k$. Similarly, in structure B, let $0 \equiv r_0 < r_1 < r_2 \ldots < r_{q-1} \equiv r$. Then traffic class *j* uses links *k*, *q* and $q + k$ if $r_{k-1} + 1 \le j \le r_k$. By (2.12), the generating functions for structures A and B with the CS policy are, respectively,

$$G(z) = \frac{1}{1-z_p} \prod_{k=1}^{p-1} \frac{\exp\left(\sum_{j=r_{k-1}+1}^{r_k} \rho_j z_k^{a_{kj}} z_p^{a_{pj}} \right)}{1-z_k} \qquad (10.1)$$

and

$$G(z) = \frac{1}{1-z_q} \prod_{k=1}^{q-1} \frac{\exp\left(\sum_{j=r_{k-1}+1}^{r_k} \rho_j z_k^{a_{kj}} z_q^{a_{qj}} z_{q+k}^{a_{q+k,j}} \right)}{(1-z_k)(1-z_{q+k})} \;. \qquad (10.2)$$

In (10.1), if we fix z_p, then each term within the product becomes independent of others and requires one-dimensional inversion. The overall inversion thereby is two-dimensional. In (10.2) if we fix z_q, then each term within the product is independent of the others and requires two-dimensional inversion. The overall inversion thereby is three-dimensional. As mentioned in Section 3.1, for the UL and GM sharing policies the inversion dimension would be either the same or one more than for the CS policy.

In the rest of this section we provide numerical examples for the tree networks (structure A). We start with the single-rate tree network example on p. 235 of Mitra (1987) using the CS policy. Here $a_{ij} = 1$ whenever it is non-zero, $r_k = k$, $r = 7$ and $p = 8$. The capacity vector is $K = (30,30,20,20,15,15,15,134)$. Equation (10.1) becomes

$$G(\mathbf{z}) = \frac{1}{1-z_8} \prod_{k=1}^{7} \frac{\exp(\rho_k z_k z_8)}{1-z_k} \quad . \tag{10.3}$$

Here inversion with respect to z_k for $k = 1,2,...,7$ may be done explicitly to get

$$g(K_1, K_2, \ldots, K_7, z_8) = \frac{1}{1-z_8} \prod_{k=1}^{7} \sum_{i=0}^{K_k} \frac{(\rho_k z_8)^i}{i!} \quad . \tag{10.4}$$

Starting from (10.4), we have a simple one-dimensional inversion with respect to z_8, which is readily done in a fraction of a second. The results are displayed in Table 7. We see that our results are between the lower and upper bounds determined by Mitra (1987) in each case. Interestingly, the results are very close to the mean of the two bounds, which Mitra suggested as an estimate. Indeed, our results agree to within Mitra's displayed accuracy of 10^{-4} in each case.

In Table 7 we also show a more challenging example where each capacity parameter K_i and each traffic load parameter p_i are multiplied by 100; i.e.,

$$K = (3000,\ 3000,\ 2000,\ 2000,\ 1500,\ 1500,\ 1500,\ 13400)$$

and ρ_j are as shown in Table 7. Using truncation we can solve even this larger example in a few minutes. it is interesting to note that the normalization constants involved in this case are of the order 10^{6483} and much larger than the upper limit 10^{307} allowed by the computer used. However, numerical overflow is avoided through scaling and by storing only the logarithms of the normalization constants. Special care also was needed in computing the quantity $\sum_{i=0}^{K_k} \frac{(\rho_k z_8)^i}{i!}$ appearing in (10.4) which causes numerical overflow in the large example for many values of k and z_8. Let,

$$s = \sum_{i=0}^{K_k} a_i \quad \text{where} \quad a_i = \frac{(\rho_k z_8)^i}{i!} \quad . \tag{10.5}$$

Let j be such that $|a_j| \geq |a_i|$ for $i = 0,1,\ldots,K_k$. It can be seen that $j \approx \min(K_k, \rho_k |z_8|)$. Now we can write

$$s = a_j \left[\sum_{i=0}^{j} \frac{a_{j-i}}{a_j} + \sum_{i=1}^{K_k - j} \frac{a_{j+i}}{a_j} \right]$$

and

$$\ln S = \ln a_j + \ln \left[1 + \sum_{i=1}^{j} \frac{a_{j-i}}{a_j} + \sum_{i=1}^{K_k - j} \frac{a_{j+i}}{a_j} \right] . \tag{10.6}$$

Computation using equation (10.6) avoids numerical overflow problems.

We now consider more general tree models with non-CS policies and multiple rates, for which existing methods do not apply. We allow more than one class to use a non-common resource and the requirements of each class for the two resources not to be identical.

Our specific example has 6 resources, with the sixth resource being the common resource. The capacity vector is $\mathbf{K} = (15,25,25,30,20,90)$. Our example has 15 classes, with class j using resources $\lceil j/3 \rceil$ and 6 (possibly at a 0 level), where $\lceil x \rceil$ is the least integer greater than or equal to x (i.e., classes 1,2 and 3 use resources 1 and 6, classes 4,5 and 6 use resources 2 and 6, etc.).

class		Mitra example			larger example	
j	ρ_j	lower bound	inversion	upper bound	ρ_j	Inversion
1	35	0.2201	0.2207492	0.2214	3500	0.14490125
2	30	0.1333	0.1340674	0.1347	3000	0.0297425
3	25	0.2801	0.2807294	0.2813	2500	0.2018138
4	17	0.0880	0.0888748	0.0896	1700	0.0287005
5	20	0.3302	0.3307832	0.3313	2000	0.2516581
6	15	0.1805	0.1822008	0.1828	1500	0.0339648
7	9	0.0257	0.0266322	0.0274	900	0.0287005

Table 7. Blocking probabilities in the single-rate tree network with the CS policy.

The specific offered loads and requirements for each class are given in Table 8. We consider both the CS and UL sharing policies. The limits for the UL policy are also given in Table 8. (These are not used with the CS policy.) The blocking probability for each class with each sharing policy is given in Table 8.

model parameters						blocking probabilities	
j	ρ_j	$a_{\lceil j/3 \rceil, j}$	$L_{\lceil j/3 \rceil, j}$	$a_{6,j}$	$L_{6,j}$	CS	UL
1	10	1	10	0	0	0.328927	0.333529
2	10	0	0	1	15	0.004393	0.036827
3	10	1	10	1	15	0.331579	0.333777
4	5	1	10	1	10	0.109389	0.104160
5	5	2	15	2	15	0.219572	0.225724
6	5	2	15	1	10	0.216223	0.225384
7	4	1	8	1	8	0.112727	0.114086
8	4	3	16	3	16	0.326724	0.330792
9	4	3	16	1	10	0.320927	0.330258
10	3	1	7	1	7	0.061176	0.025887
11	3	4	15	4	15	0.236906	0.347366
12	3	4	15	1	8	0.226213	0.347187
13	2	1	6	1	6	0.075576	0.046674
14	2	5	13	5	13	0.433225	0.465534
15	2	5	13	1	5	0.422868	0.465049

Table 8. Blocking probabilities in a multi-rate tree network with the CS and UL sharing policies.

We obtain the generating function for CS from (10.1). It is

$$G(\mathbf{z}) \;=\; \exp\left[\sum_{j=1}^{15} \rho_j z_{\lceil j/3 \rceil}^{a_{\lceil j/3 \rceil ,j}} z_6^{a_{6j}}\right] \Big/ \prod_{i=1}^{6} (1 - z_i) \; . \tag{10.7}$$

We employ conditional decomposition to reduce the dimension from 6 to 2; i.e., for any fixed value of z_6, the generating function can be represented as a product of 5 factors with z_i appearing only in the i^{th} factor. We obtain the generating function for the UL policy from (2.17):

$$G(\mathbf{z},\mathbf{y}) \;=\; \frac{\exp\left(\sum_{j=1}^{15} \rho_j y_j z_{\lceil j/3 \rceil}^{a_{\lceil j/3 \rceil ,j}} z_6^{a_{6,j}}\right)}{\prod_{i=1}^{6} (1 - z_i) \prod_{j=1}^{15} (1 - y_j)} \; . \tag{10.8}$$

Conditional decomposition reduces the dimension from 21 to 3. As with the CS policy, we first invert z_6, then we invert the other z_i variable in each factor. For any fixed values of the two z variables, the generating function can be represented as a product of 15 factors with y_j appearing only in the j^{th} factor.

The computation of all blocking probabilities took several seconds. We can increase the numbers of classes, resources and capacities each by factors of ten and still carry out the computations in several minutes using truncation.

11. Conclusion

In this paper we have developed a new algorithm for a family of models that is the natural generalization of both loss networks and infinite-server variants of the resource-sharing models. We considered generalizations of loss networks allowing the UL and GM sharing policies as well as the customary CS sharing policy. Equivalently, we considered the generalization of the (infinite-server variant of the) resource-sharing model with multiple resources. In Choudhury, Leung and Whitt (1994a,b) we also treat state-dependent arrivals and batch arrivals.

Of course, the idea of such general models is not new; e.g., Jordan and Varaiya (1991) discuss multi-resource resource-sharing models, and Chuah (1993) develops a recursive algorithm for the UL policy with Poisson sources and one or two resources. The principal contribution here is a new effective algorithm for solving these models.

Our algorithm has been shown to be effective on several numerical examples. To validate our algorithm on small models, we developed the direct algorithm in Section 6. In addition, we implemented the recursive algorithm of Kaufman (1981) and Roberts (1981) the uniform asymptotic approximation of Mitra and Morrison (1993). In all cases that these algorithms applied, the inversion approach agreed to an accuracy of several digits. Our algorithm also has a built-in accuracy check which can independently confirm accuracy.

A great appeal of the numerical inversion algorithm is that it is very general. It not only applies to different kinds of product-form models through numerical inversion of the generating functions of the normalization constants, but it also applies to many other models, where the quantity of interest is represented via a transform. See Choudhury, Lucantoni and Whitt (1994b) for a review of recent applications of numerical inversion to queueing models.

REFERENCES

ABATE J. and WHITT W. 1992a. The Fourier-Series method for Inverting Transforms of Probability Distributions. *Queueing Systems* **10,** 5-88.

ABATE, J. and WHITT, W. 1992b. Numerical Inversion of Probability Generating Functions. *Opns. Res. Letters* **12,** 245-251.

AEIN, J. M. 1978. A Multi-User-Class, Blocked-Calls-Cleared Demand Access Mode. *IEEE Trans. Commun.* **COM-26,** 378-385.

AEIN, J. M. and KOSOVYCH, O. S. 1977. Satellite Capacity Allocation. *Proc. IEEE* **65,** 332-342.

ASH, G. R., and HUANG, B. D. 1993. An Analytical Model for Adaptive Routing Networks. *IEEE Trans. Commun.* **41,** 1748-1759.

BERTOZZI, A. and MCKENNA, J. 1993. Multidimensional Residues, Generating Functions, and Their Application to Queueing Networks. *SIAM Review* **35,** 239-268.

BURMAN, D.Y., LEHOCZKY, J.P. and LIM, Y. 1984. Insensitivity of Blocking Probabilities in a Circuit Switching Network. *J. Appl. Prob.* **21,** 850-859.

BUZEN, J. P. 1973. Computational Algorithms for the Closed Queueing Networks with Exponential Servers. *Commun. ACM* **16,** 527-531.

CHOUDHURY, G. L., LEUNG, K. K. and WHITT, W. 1993. Calculating Normalization Constants of Closed Queueing Networks by Numerically Inverting Their Generating Functions. *J. ACM,* to appear.

CHOUDHURY, G. L., LEUNG, K. K. and WHITT, W. 1994a. An Inversion Algorithm for Loss Networks with State-Dependent Arrivals. in preparation.

CHOUDHURY, G. L., LEUNG, K. K. and WHITT, W. 1994b. Loss Models with Batch Arrivals. in preparation.

CHOUDHURY, G. L., LUCANTONI, D. M. and WHITT, W. 1994a. Multidimensional Transform Inversion With Applications to the Transient M/G/1 Queue. *Ann. Appl. Prob.* to appear.

CHOUDHURY, G. L., LUCANTONI, D. M. and WHITT, W. 1994b. Numerical Transform Inversion to Analyze Teletraffic Models. *The Fundamental Role of Teletraffic in the Evolution of Telecommunications Networks, Proceedings of the 14th International Teletraffic Congress,* J. Labetoulle and J. W. Roberts (eds.), Elsevier, Amsterdam, 1b, 1043-1052.

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

CHUNG, S.-P., and ROSS, K. 1993. Reduced Load Approximations for Multirate Loss Networks. *IEEE Trans. Commun.* **41,** 1222-1231.

CONWAY, A. E. and PINSKY, E. 1992. A Decomposition Method for the Exact Analysis of Circuit-Switched Networks. *Proceedings of IEEE Infocom '92,* 996-1003.

DELBROUCK, L. W. N. 1983. On the Steady-State Distribution in a Service Facility Carrying Mixtures of Traffic with Different Peakedness Factors and Capacity Requirements. *IEEE Trans. Commun.* **31,** 1209-1211.

DZIONG, Z., and ROBERTS, J. W. 1987. Congestion Probabilities in a Circuit-Switched Integrated Services Network. *Perf. Eval.* **7,** 267-284.

EVANS, S. P. 1991. Optimal Bandwidth Management and Capacity Provision in a Broadband Network Using Virtual Paths. *Perf. Eval.* **13,** 27-43.

GAUJAL, B., GREENBERG, A. and NICOL, D. 1993. A Sweep Algorithm for Massively Parallel Simulation of Circuit-Switched Networks. *J. Parallel and Distributed Computing.*

GREENBERG, A., NICOL, D. and LUBACHEVSKY, B. 1992. MIMD Parallel Simulation of Circuit-Switched Communication Networks. *Proceedings 1992 Winter Simulation Conference.*

JORDAN, S. and VARAIYA, P. P. 1991. Throughput in Multiple Service, Multiple Resource Communication Networks. *IEEE Trans. Commun.* **39,** 1216-1222.

KAMOUN, F., and KLEINROCK, L. 1980. Analysis of Shared Finite Storage in a Computer Network Node Environment Under General Traffic Conditions. *IEEE Trans. Commun.* **COM-28,** 992-1003.

KAUFMAN, J. S. 1981. Blocking in a Shared Resource Environment. *IEEE Trans. Commun.* **COM-29,** 1474-1481.

KAUFMAN, J. S. and REGE, K. M. 1993. Blocking in a Shared Resource Environment with Batched Poisson Arrival Processes. AT&T Bell Laboratories, Holmdel, NJ.

KELLY, F. P. 1979. *Reversibility and Stochastic Networks,* Wiley, New York.

KELLY, F. P. 1985. Stochastic Models of Computer Communication Systems. *J. R. Statist. Soc. B* **47,** 379-395.

KELLY, F. P. 1986. Blocking Probabilities in Large Circuit-Switched Networks. *Adv. Appl. Prob.* **18,** 473-505.

KELLY, F. P. 1991. Loss Networks. *Ann. Appl. Prob.* **1,** 319-378.

KOGAN, Y. 1989. Exact Analysis for a Class of Simple, Circuit-Switched Networks with Blocking. *Adv. Appl. Prob.* **21,** 952-955.

LABOURDETTE, J. P., HART, G. W. 1992. Blocking Probabilities in Multitraffic Loss Systems: Insensitivity, Asymptotic Behavior, and Approximations. *IEEE Trans. Commun.* **40,** 1355-1366.

LAM, S. S. 1977. Queueing Networks with Population Size Constraints. *IBM J. Res. Develop.* 21, 370-378.

LAM, S. S. and LIEN, Y. L. 1983. A Tree Convolution Algorithm for the Solution of Queueing Networks. *Commun. ACM* 26, 203-215.

MELAMED, B. and WHITT, W. 1990. On Arrivals That See Time Averages. *Opns. Res.* **38,** 156-172.

MITRA, D. 1987. Asymptotic Analysis and Computational Methods for a Class of Simple, Circuit-Switched Networks with Blocking. *Adv. Appl. Prob.* **19,** 291-239.

MITRA, D., GIBBENS, R. J., and HUANG, B. D. 1993. State-Dependent Routing on Symmetric Loss Networks with Trunk Reservations - I. *IEEE Trans. Commun.* **41,** 400-411.

MITRA, D., and MORRISON, J. A. 1993. Erlang Capacity and Uniform Approximations for Shared Unbuffered Resources. AT&T Bell Laboratories, Murray Hill, New Jersey.

MORRISON, J. A. 1993. Blocking Probabilities for Multiple Class Batched Poisson Arrivals to a Shared Resource. AT&T Bell Laboratories, Murray Hill, NJ.

PINSKY, E. and CONWAY, A. E. 1992. Computational Algorithms for Blocking Probabilities in Circuit-Switched Networks. *Ann. Opns. Res.* 35, 31-41.

REIMAN, M. I. 1991. A Critically Loaded Multiclass Erlang Loss System. *Queueing Systems* **9,** 65-82.

REISER, M. and KOBAYASHI, H. 1975. Queueing Networks with Multiple Closed Chains: Theory and Computational Algorithms. *IBM J. Res. Dev.* 19, 283-294.

ROBERTS, J. W. 1981. A Service System with Heterogeneous User Requirements. *Perf. of Data Commun. Systems and their Applications,* G. Pujolle (Ed.), North-Holland Publishing, 423-431.

ROSS, K. W. and TSANG, D. H. K. 1989a. The Stochastic Knapsack Problem. *IEEE Trans. Commun.* **37,** 740-747.

ROSS, K. W. and TSANG, D. H. K. 1989b. Optimal Circuit Access Policies in an ISDN

Environment: A Markov Decision Approach. *IEEE Trans. Commun.* **37,** 934-939.

ROSS, K. W., and TSANG, D. 1990. Teletraffic Engineering for Product-Form Circuit-Switched Networks. *Adv. Appl. Prob.* **22,** 657-675.

ROSS, K. W. and WANG, J. 1992. Monte Carlo Summation Applied to Product-Form Loss Networks. *Prob. Engr. Inf. Sci.* **6,** 323-348.

TSANG, D., and ROSS, K. W. 1990. Algorithms to Determine Exact Blocking Probabilities for Multirate Tree Networks. *IEEE Trans. Commun.* **38,** 1266-1271.

VAN DOORN, E. A. and PANKEN, J. F. M. 1993. Blocking Probabilities in a Loss System with Arrivals in Geometrically Distributed Batches and Heterogeneous Service Requirements. *IEEE/ACM Trans. Networking* **1,** 664-667.

VAN DE VLAG, H. A. B. and AWATER, G. A. 1994. Exact Computation of Time and Call Blocking Probabilities in Multi-Traffic Circuit-Switched Networks. *Proceedings IEEE Infocom '94,* 56-65.

WHITT, W. 1980. Continuity of Generalized Semi-Markov Processes. *Math. Opns. Res.* 5, 494-501.

WHITT, W. 1985. Blocking When Service is Required from Several Facilities Simultaneously *AT&T Tech. J.* 64, 1807-1855.

WIMP, J. 1981. *Sequence Transformations and Their Applications,* Academic Press, New York.