Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

P.R. de Waal

A constrained optimisation problem in a processor sharing queue

Department of Operations Research, Statistics, and System Theory Report BS-R9025 September

---

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Research (N.W.O.).

A Constrained Optimisation Problem in a Processor Sharing Queue

Peter de Waal
Centre for Mathematics and Computer Science
P.O.Box 4079, 1009 AB Amsterdam, The Netherlands

In this paper we discuss a Processor Sharing queueing model for a Stored Program Controlled telephone exchange. The model incorporates the effects of both call requests and operator tasks on the load of the processor. Newly arriving call requests and operator tasks can either be admitted or rejected and for this decision the state of the queue at the moment of arrival is available as information. To guarantee a high level of service, we formulate a constrained optimisation problem. Two types of access control, viz. partitioning policies and sharing policies are considered. The optimisation problem is solved for partitioning policies and the performance of both types of policies are compared.

1980 Mathematics Subject Classification: 60K25, 90B22.

Keywords & Phrases: queueing networks, constrained optimisation, product form.

1 INTRODUCTION

In this chapter we shall discuss a Processor Sharing queueing model for a *Stored Program Controlled (SPC)* telephone switch. The Central Control Unit (CCU) of an SPC is a processor module running an operating system that is specially designed to maintain telephone traffic. Specifically this means that, apart from serving call requests, the CCU has to use part of its capacity to perform operator tasks. *Operator tasks* are issued by the operator and they are not directly related to call request processing. As examples one can think of administration and test procedures. To guarantee proper operation of the switch, a part of the processor capacity has to be available to perform operator tasks at any time. What remains of the capacity may be used to handle call requests. This model is introduced to analyse the impact on the load of the processor caused by call requests and operator tasks.

If the switch is only lightly loaded by call requests, then we can assign part of the remaining idle capacity to take care of operator tasks. If the switch is operating in an overload situation, that we assume is caused by call requests, then clearly all of the call handling capacity has to be assigned for this purpose. In such a situation operator tasks will receive only their guaranteed minimal portion of processor capacity.

In the model that we shall discuss the processor is represented by a Processor Sharing queue. This service discipline is an appropriate tool in the modelling of the time sharing operating system of a computer (cf. KLEINROCK [7]). We consider an overload control algorithm that is allowed to reject or admit call requests and operator tasks. The information that will be

available for the control algorithm consists of the number of call requests and the number of operator tasks that are in service. The main objective of the algorithm is to regulate the admission to the processor to maximise the number of completed call requests, under the constraints that the delay a request encounters in waiting and serving is limited and that a certain part of the processor's capacity is always available to deal with operator tasks. We can translate this objective into the terminology of queueing theory as: maximise the throughput of call requests while maintaining an upper bound on their mean sojourn times and at the same time guaranteeing a lower bound on the throughput of operator tasks. In this formulation the impatience of subscribers is thus implicitly accounted for by the constraint on the mean sojourn times of call requests. We have chosen this approach to model impatience, since explicit modelling would require the derivation of the sojourn time distribution of call requests in a Processor Sharing queue which also handles operator tasks. Although quite recently progress has been made in this area (cf. VAN DEN BERG [1]), it is not yet clear how these results can be put to advantage in optimal control problems. Alternative approaches to implicitly account for impatience shall be mentioned in the references later in this section.

In this chapter we shall restrict our attention to two classes of admission policies. In both classes the decision to admit new call requests and operator tasks is non-randomised, i.e. requests and tasks are either rejected or admitted. If a task or request is admitted, it joins the queue for service, otherwise it is lost. The first class of admission policies, called *partitioning policies*, is implemented by two separate thresholds, one for call requests and one for operator tasks. If the number of call requests in service reaches the corresponding threshold, a new request is rejected until the number of requests in service decreases by a service completion. An analogous algorithm is employed for operator tasks. The second class of admission policies that we consider, are called *sharing policies*. A sharing policy uses a pool that contains c permission units (*p.u.'s*) for some $c \in \mathbb{N}$. If a call request arrives at the queue, it can be admitted only if at least c_1, $c_1 \in \mathbb{N}$, of p.u.'s are available in the pool. If these p.u.'s are available, then they are removed from the pool and the request is allowed to enter the queue (cf. Figure 1). After the call request has completed its service, the p.u.'s of this request are put back into the pool. An analogous strategy is used for operator tasks, though c_2, the number of p.u.'s needed for admission of a task, may differ from c_1. This control was implemented to allow "sharing" of the processor capacity, i.e. when the demand for service of call requests is low, then operator tasks are allowed to use part of the call handling capacity and vice versa.

In this chapter we shall discuss the constrained control problem, that was introduced in the beginning of this section, for both classes of admission policies. We prove that the problem can be solved for partitioning policies. Although we have not been able to solve the problem for sharing policies, we shall discuss the performance of such policies for several parameter settings.

Constrained optimisation problems for queues have gained increasing interest of researchers during the last five years. In ROSS [15] finite state Markov Decision Processes are considered with a reward and cost structure. The objective is to maximise the average return with a constraint on the average costs. In HORDIJK AND SPIEKSMA [4] an optimisation problem with a constraint on the average cost for a one-dimensional queueing system is discussed. They show that for rather general assumptions on the cost and reward structure the optimal control randomises in exactly one state. In MA AND MAKOWSKI [10] a constrained optimal flow control problem is solved with Lagrangian methods. Again the optimal control is shown to be

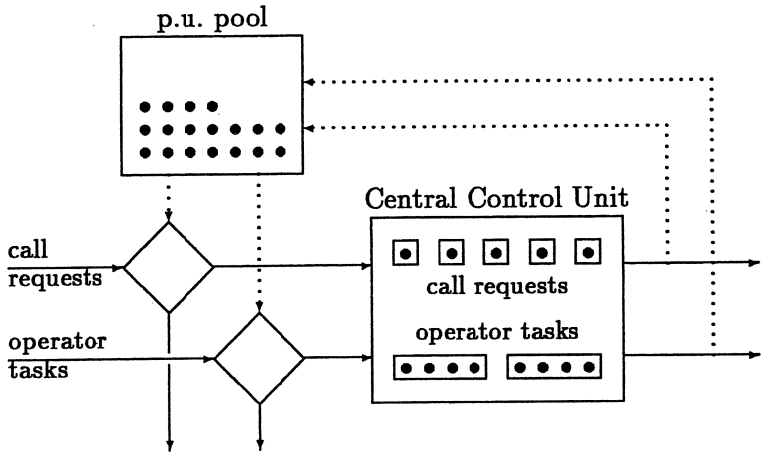

FIGURE 1. A Processor Sharing queue with a pool of permission units.

randomised in exactly one state. A well known reference on constrained optimal flow control is ROBERTAZZI AND LAZAR [14]. They consider the problem of maximising the throughput of a one-dimensional queue under a constraint on the average delay. The optimal control is shown to be of a window flow type and again it is randomised in exactly one state. In NAIN AND ROSS [11] a queueing model is discussed for multiplexing heterogeneous Poisson arrival streams onto a single communication channel. The optimisation problem is to minimise a linear combination of the average delays, while at the same time subjecting the average delay for one stream to a hard constraint. This constraint is an alternative approach to account for impatience of customers implicitly. The optimal multiplexing policy is shown to be a randomised modification of the μc-rule. In NAIN AND ROSS [12] a similar system is considered with renewal arrival streams. To minimise a linear combination of the average queue lengths with a hard constraint on the average queue length of one arrival stream, the optimal multiplexing policy is shown to be a randomised modification of a static-priority rule. The performance analysis of partitioning and sharing policies is related to the analysis of resource sharing in KAUFMAN [5]. He derives a product form for the equilibrium distribution of a queueing model with sharing policies and presents a one-dimensional recursion for the computation of performance measures. In FOSCHINI AND GOPINATH [3] an optimal control problem for a queueing system with two processors and a common waiting room is presented. They show that the control that minimises a weighted sum of the idle times of the servers, is a combination of a partitioning and a sharing policy.

This chapter is organised as follows. In Section 2 we present a description of the queueing model and formulate the control problems for the partitioning and sharing policies. Partitioning policies are discussed in Section 3. In Section 3.1 we derive monotonicity properties of performance measures, that are used in Section 3.2 to establish conditions for the existence of an optimal policy. A design algorithm for the optimal policy is discussed in Section 3.3. We address the performance of sharing policies in Section 4. The chapter is concluded with some numerical examples in Section 5.

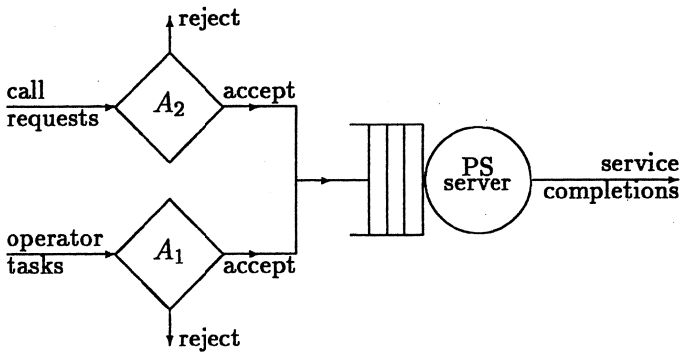

FIGURE 2. A Processor Sharing queueing model.

2 DESCRIPTION OF THE QUEUEING MODEL

Consider a Processor Sharing queue with two independent Poisson arrival processes. Customers of class 1 represent the call requests and they arrive with rate λ_1. Operator tasks are referred to as class 2 customers and they arrive with rate λ_2 (see Figure 2).

The service requirements for customers of class i, are independent and exponentially distributed with mean $1/\mu_i$, $i = 1, 2$. We define the workloads $\rho_i = \lambda_i/\mu_i$, $i = 1, 2$. The server works with constant service rate according to the Processor Sharing discipline, i.e. at any time each customer receives the same amount of service as any other customer.

The queueing process is a continuous time Markov process $\mathcal{X} = (X_t, t \geq 0)$ on the state space $\mathcal{S} = \mathbb{N}^2$, where a state is represented by a population vector $\mathbf{m} = (m_1, m_2)$ corresponding to a population of m_i customers of class i, $i = 1, 2$. This means that we do not distinguish different customers within one class, but only keep track of the population of both classes.

Upon arrival to the queue customers can either be admitted or rejected according to a stationary admission policy. In accordance with the notation in DE WAAL AND VAN DIJK [16] we can represent such a stationary admission policy by two arrival probability functions A^1, $A^2 : \mathbb{N}^2 \to \{0, 1\}$. If the queue population is $\mathbf{m} \in \mathbb{N}^2$, then a new arriving customer of class i is rejected or admitted if $A^i(\mathbf{m}) = 0$ or 1, respectively. Admitted customers join the queue until completion of service, while rejected customers are assumed lost without further influencing the future of the arrival and queueing processes.

In the remaining sections of this chapter we shall restrict our attention to admission policies that are described by coordinate convex subsets of $\mathbb{N}^2$ (cf. LAM [8] and Example 2.6 in DE WAAL AND VAN DIJK [16]). If we let $\mathcal{C} \subset \mathbb{N}^2$ denote a coordinate convex set, then we define the stationary policy $f_{\mathcal{C}}$ as the admission policy that restricts the state space to $\mathcal{C}$. In words this means that customers are admitted only if this does not result in a population vector outside of the set $\mathcal{C}$, so

$$A^i(\mathbf{m}) = 1(\mathbf{m} + \mathbf{e}_i \in \mathcal{C}), \qquad \mathbf{m} \in \mathbb{N}^2,\ i = 1, 2. \tag{1}$$

The set $\mathcal{C}$ is called the admission region of $f_{\mathcal{C}}$. We do not exercise any control on departures of customers. In this chapter we shall make a restriction to two subclasses of coordinate convex sets in $\mathbb{N}^2$.

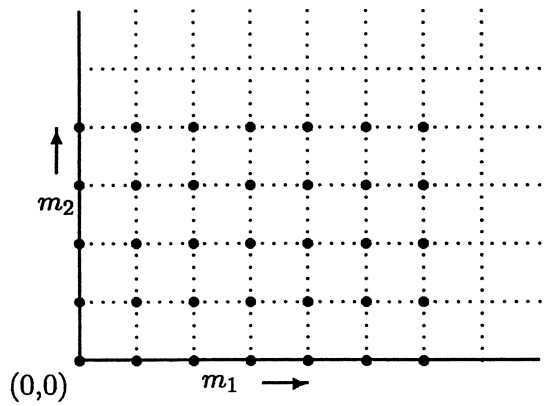

FIGURE 3. State space for a partitioning policy.
Thresholds $M_1 = 6$ and $M_2 = 4$.

DEFINITION 2.1 (PARTITIONING POLICIES)
If for some M_1, $M_2 \in \mathbb{N} \cup \{\infty\}$

$$\mathcal{C} = \{\, \mathbf{m} \in \mathbb{N}^2 \mid 0 \le m_1 \le M_1, 0 \le m_2 \le M_2 \,\}, \tag{2}$$

then $f_{\mathcal{C}}$ *is called a* partitioning policy with thresholds M_i for class i, $i = 1, 2$. *The set* $\mathcal{C}$ *is called a* partition *and the set of partitioning policies is denoted as* $\mathcal{F}_P$.

The state space of the queueing process $\mathcal{X}$ restricted to a partition with thresholds $M_1 = 6$ and $M_2 = 4$ is depicted in Figure 3. The effect of such a policy is that the separate admission policies for both classes become "independent", i.e. the population of one class does not affect the admission probability for the other class. One can also say that the admission policy for one class uses only partial state information for its decision, viz. it uses only the population size of its corresponding class.

The second class of coordinate convex admission regions corresponds to triangles in $\mathbb{N}^2$.

DEFINITION 2.2 (SHARING POLICIES)
If for some c_1, $c_2 \in \mathbb{N}$, $c \in \mathbb{N} \cup \{\infty\}$

$$\mathcal{C} = \{\, \mathbf{m} \in \mathbb{N}^2 \mid 0 \le c_1 m_1 + c_2 m_2 \le c \,\}, \tag{3}$$

then $f_{\mathcal{C}}$ *is called a* sharing policy *with* weight factors c_i *for class* i, $i = 1, 2$, *and* poolsize c. *The set of sharing policies is denoted as* $\mathcal{F}_S$.

An example of the state space of a sharing policy with weight factors $c_1 = 1$, $c_2 = 2$ and poolsize $c = 8$ is depicted in Figure 4.

The terms partitioning and sharing were used in KAUFMAN [5] in the context of memory allocation policies for computers (see also FOSCHINI AND GOPINATH [3]).

Since both the partitioning and the sharing policies allow infinite state spaces, we need some conditions that guarantee that the controlled queueing process is ergodic. It is easy to check that for ergodicity the following conditions are sufficient.

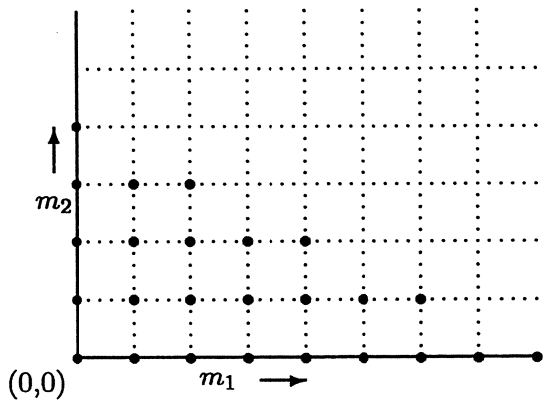

FIGURE 4. State space for a sharing policy.
Weight factors $c_1 = 1$, $c_2 = 2$ and poolsize $c = 8$.

CONDITION 2.3 (ERGODICITY CONDITIONS)
Let $\mathcal{C}$ be the admission region of policy $f_{\mathcal{C}}$. The queueing process that is restricted to the set $\mathcal{C}$ is ergodic if all of the following conditions are true.

(i) If $\limsup_{\mathbf{m}\in\mathcal{C}} m_1 = \infty$, *then* $\rho_1 < 1$.

(ii) If $\limsup_{\mathbf{m}\in\mathcal{C}} m_2 = \infty$, *then* $\rho_2 < 1$.

(iii) If $\limsup_{\mathbf{m}\in\mathcal{C}} m_1 = \limsup_{\mathbf{m}\in\mathcal{C}} m_2 = \infty$, *then* $\rho_1 + \rho_2 < 1$.

(iv) If $\mathcal{C}$ is a finite set, then the queueing process is ergodic for all values of λ_1, λ_2, μ_1, μ_2.

We can now proceed with the formulation of the performance measures throughput, expected sojourn time and expected queue length. Let $\mathcal{C}$ be a coordinate convex subset of $\mathbb{N}^2$. We introduce for $i = 1, 2$, $n \in \mathbb{N}$, $X_0 \in \mathbb{N}^2$, $t \geq 0$ and $h > 0$ the random variables

$\mathbf{S}_i(n, \mathcal{C}, X_0)$ The sojourn time or residence time in the queue of the n-th admitted customer of class i under admission policy $f_{\mathcal{C}}$ when the initial state is X_0.

$\mathbf{L}_i(t, \mathcal{C}, X_0)$ The number of customers of class i in the queue at time t under policy $f_{\mathcal{C}}$ when the initial state is X_0.

$\mathbf{T}_i(t, t+h, \mathcal{C}, X_0)$ The number of customers of class i that depart from the queue in the interval $[t, t+h)$ under policy $f_{\mathcal{C}}$ when the initial state is X_0.

If we assume that the ergodicity conditions are satisfied, then we can define

$$S_i(\mathcal{C}) = \lim_{n\to\infty} E[\mathbf{S}_i(n, \mathcal{C}, X_0)],$$

$$L_i(\mathcal{C}) = \lim_{t\to\infty} E[\mathbf{L}_i(t, \mathcal{C}, X_0)],$$

$$T_i(\mathcal{C}) = \lim_{h \downarrow 0} \lim_{t \to \infty} \frac{E[\mathbf{T}_i(t, t+h, \mathcal{C}, X_0)]}{h}.$$

These quantities are well defined and independent of the initial state X_0. It is well known from literature that these three performance measures can be expressed in the equilibrium distribution $\pi_{\mathcal{C}}$ of process $\mathcal{X}_{\mathcal{C}}$ (cf. REISER AND KOBAYASHI [13]). We know from LAM [8, page 373] and DE WAAL AND VAN DIJK [16, Theorem 2.9] that the equilibrium distribution $\pi_{\mathcal{C}}(\mathbf{m})$ of state $\mathbf{m} = (m_1, m_2)$ is

$$\pi_{\mathcal{C}}(\mathbf{m}) = G(\mathcal{C})\, 1(\mathbf{m} \in \mathcal{C}) \binom{m_1 + m_2}{m_1} \rho_1^{m_1} \rho_2^{m_2}, \qquad \mathbf{m} \in \mathbb{N}^2, \tag{4}$$

where $G(\mathcal{C})$ is the normalisation constant that guarantees that $\sum_{\mathbf{m} \in \mathcal{C}} \pi_{\mathcal{C}}(\mathbf{m}) = 1$. The expressions for the *throughput* $T_i(\mathcal{C})$ and the *expected queue length* $L_i(\mathcal{C})$ for class i under admission policy $f_{\mathcal{C}}$ are

$$T_i(\mathcal{C}) = \sum_{\mathbf{m} \in \mathcal{C}} \pi_{\mathcal{C}}(\mathbf{m})\, \lambda_i\, 1(\mathbf{m} + \mathbf{e}_i \in \mathcal{C}), \qquad i = 1, 2, \tag{5}$$

$$= \sum_{\mathbf{m} \in \mathcal{C}} \pi_{\mathcal{C}}(\mathbf{m}) \frac{m_i \mu_i}{m_1 + m_2} 1(m_1 + m_2 > 0), \qquad i = 1, 2, \tag{6}$$

$$L_i(\mathcal{C}) = \sum_{\mathbf{m} \in \mathcal{C}} \pi_{\mathcal{C}}(\mathbf{m}) m_i, \qquad i = 1, 2. \tag{7}$$

The *expected sojourn time* S_i *for class* i can be expressed in T_i and L_i through Little's law (cf. LITTLE [9]):

$$L_i(\mathcal{C}) = T_i(\mathcal{C}) S_i(\mathcal{C}), \qquad i = 1, 2. \tag{8}$$

With these definitions we are now all set to introduce the constrained control problems.

DEFINITION 2.4 *Let S and T be fixed constants in $\mathbb{R}_+$ satisfying $S \geq 1/\mu_1$ and $T \leq \lambda_2$. The problems (P_P) and (P_S) are defined as*

(P_P): *maximise $T_1(\mathcal{C})$ over $f_{\mathcal{C}} \in \mathcal{F}_P$ under the constraints $S_1(\mathcal{C}) \leq S$ and $T_2(\mathcal{C}) \geq T$. A partition $\mathcal{C}$ is called* optimal *for (P_P) if it satisfies the constraints on S_1 and T_2 and attains the maximum for T_1.*

(P_S): *maximise $T_1(\mathcal{C})$ over $f_{\mathcal{C}} \in \mathcal{F}_S$ under the constraints $S_1(\mathcal{C}) \leq S$ and $T_2(\mathcal{C}) \geq T$. A sharing policy $f_{\mathcal{C}}$ is called* optimal *for (P_S) if it satisfies the constraints on S_1 and T_2 and attains the maximum for T_1.*

The condition $S \geq 1/\mu_1$ is to guarantee that the constrained problems are not trivial, since the mean sojourn time of a class 1 customer can never be smaller than its mean service time. For the same reason the condition $T \leq \lambda_2$ was introduced, since the throughput of class 2 customers can never be larger than the arrival rate.

We shall call a policy $f_{\mathcal{C}} \in \mathcal{F}_P$ *feasible for* (P_P) if the queueing process $\mathcal{X}$ is ergodic under $f_{\mathcal{C}}$ and $S_1(\mathcal{C}) \leq S$ and $T_2(\mathcal{C}) \geq T$. An analogous definition for feasibility of sharing policies is used.

We conclude this section with a remark about the choice of the exponential service time distributions. We know from the references that were already mentioned — LAM [8] and DE WAAL AND VAN DIJK [16] — that the product form equilibrium distribution remains valid even for general service time distributions. It is also known (cf. for instance KELLY [6] and REISER AND KOBAYASHI [13]) that the throughput, expected queue length and sojourn time depend only on the mean of the service time distributions. As a consequence the problems (P_P) and (P_S) are in fact defined for general service time distributions, and the optimal policy depends only on the mean of the service times. A preliminary result is thus the robustness of the optimal admission policies with respect to the service time distributions. In an attempt to keep the notation simple, however, we shall restrict ourselves to the exponential case in the remainder of this chapter.

We shall address the constrained optimisation problems (P_P) and (P_S) in Sections 3 and 4, respectively.

3 PARTITIONING POLICIES

3.1 Expressions for performance measures

In this section we shall discuss the constrained control problem (P_P) for partitioning policies. We shall derive expressions for the performance measures T_1, T_2, and S_1 as functions of a partition's thresholds. We shall also establish monotonicity results for these performance measures with respect to the thresholds. These results are employed to formulate existence conditions for an optimal policy in Section 3.2 and to design a search procedure for the optimal thresholds in Section 3.3.

First we introduce some notation. For a partitioning policy $f_{\mathcal{C}} \in \mathcal{F}_P$ with thresholds M_1 and M_2 we denote $\mathcal{C}$ as $\langle M_1, M_2 \rangle$, $G(\mathcal{C}) = G(M_1, M_2)$, and for $i = 1, 2$, $T_i(\mathcal{C}) = T_i(M_1, M_2)$, $L_i(\mathcal{C}) = L_i(M_1, M_2)$ and $S_i(\mathcal{C}) = S_i(M_1, M_2)$. We allow M_1 and M_2 to be infinite. We start with the derivation of expressions for the performance measures. For this it is convenient to introduce the unnormalised equilibrium distribution $\bar{\pi} : \mathbb{N}^2 \to \mathbb{R}_+$,

$$\bar{\pi}(\mathbf{m}) = \binom{m_1 + m_2}{m_1} \rho_1^{m_1} \rho_2^{m_2}, \qquad \mathbf{m} \in \mathbb{N}^2. \tag{9}$$

From (4) the normalising constant $G(\mathcal{C})$ for a partition $\mathcal{C} = \langle M_1, M_2 \rangle$, can be expressed as

$$G(M_1, M_2) = \Big[\sum_{m_1=0}^{M_1} \sum_{m_2=0}^{M_2} \bar{\pi}(\mathbf{m}) \Big]^{-1}. \tag{10}$$

Note that $G(\mathcal{C})$ is defined only if the ergodicity conditions are satisfied, i.e. $\rho_1 < 1$ if $M_1 = \infty$, $\rho_2 < 1$ if $M_2 = \infty$ and $\rho_1 + \rho_2 < 1$ if $M_1 = M_2 = \infty$. The performance measures can now be expressed as

$$T_1(M_1, M_2) = \lambda_1 G(M_1, M_2) \sum_{m_1=0}^{M_1-1} \sum_{m_2=0}^{M_2} \bar{\pi}(m_1, m_2), \tag{11}$$

$$T_2(M_1, M_2) = \lambda_2 G(M_1, M_2) \sum_{m_1=0}^{M_1} \sum_{m_2=0}^{M_2-1} \bar{\pi}(m_1, m_2), \tag{12}$$

$$L_i(M_1, M_2) = G(M_1, M_2) \sum_{m_1=0}^{M_1} \sum_{m_2=0}^{M_2} m_i \, \bar{\pi}(m_1, m_2), \qquad i = 1, 2. \tag{13}$$

The expected sojourn time follows from Little's law (8):

$$S_i(M_1, M_2) = \frac{L_i(M_1, M_2)}{T_i(M_1, M_2)}, \qquad i = 1, 2. \tag{14}$$

Again the remark about the ergodicity conditions applies. For finite thresholds the performance measures can be computed quite easily by a recursive scheme in M_1 and M_2 (cf. REISER AND KOBAYASHI [13]). However, since a feasible partition may have one or two infinite thresholds, we must also be able to compute the measures for these cases. To accomplish this we define for $n \in \mathbb{N}$, $0 \leq x < 1$, $y \geq 0$:

$$J_1(n, x, y) = \sum_{j=0}^{n} \sum_{k=0}^{j} \binom{j}{k} \frac{y^j \, x^{j-k}}{(1-x)^{j+1-k}}, \tag{15}$$

$$J_2(n, x, y) = \sum_{j=0}^{n} \sum_{k=0}^{j+1} \binom{j+1}{k} \frac{(j+1) \, y^j \, x^{j+2-k}}{(1-x)^{j+2-k}}, \tag{16}$$

$$J_3(n, x, y) = \sum_{j=1}^{n} \sum_{k=0}^{j} \binom{j}{k} \frac{j \, y^j \, x^{j+1-k}}{(1-x)^{j+1-k}}. \tag{17}$$

The computation of these values can be implemented into an algorithm by recursion in n. Furthermore, the performance measures with infinite thresholds can be expressed in the finite sums J_1, J_2, J_3, as is illustrated in the following lemma.

LEMMA 3.1 *For all finite* $M_1, M_2 \in \mathbb{N}$

$$\begin{aligned}
T_1(\infty, M_2) &= \lambda_1 &&= \lim_{M_1 \to \infty} T_1(M_1, M_2), && \rho_1 < 1, \\
S_1(\infty, M_2) &= \frac{J_2(M_2, \rho_1, \rho_2)}{\lambda_1 J_1(M_2, \rho_1, \rho_2)} &&= \lim_{M_1 \to \infty} S_1(M_1, M_2), && \rho_1 < 1, \\
T_2(\infty, M_2) &= \lambda_2 \frac{J_1(M_2 - 1, \rho_1, \rho_2)}{J_1(M_2, \rho_1, \rho_2)} &&= \lim_{M_1 \to \infty} T_2(M_1, M_2), && \rho_1 < 1, \\
T_1(M_1, \infty) &= \lambda_1 \frac{J_1(M_1 - 1, \rho_2, \rho_1)}{J_1(M_1, \rho_2, \rho_1)} &&= \lim_{M_2 \to \infty} T_1(M_1, M_2), && \rho_2 < 1, \\
S_1(M_1, \infty) &= \frac{J_3(M_1, \rho_2, \rho_1)}{\lambda_1 J_1(M_1 - 1, \rho_2, \rho_1)} &&= \lim_{M_2 \to \infty} S_1(M_1, M_2), && \rho_2 < 1, \\
T_2(M_1, \infty) &= \lambda_2 &&= \lim_{M_2 \to \infty} T_2(M_1, M_2), && \rho_2 < 1.
\end{aligned}$$

PROOF. Examination of expressions (11)–(14) learns that we have to prove for $0 \leq x < 1$, $y \geq 0$,

$$J_1(n,x,y) = \lim_{k\to\infty} \sum_{i=0}^{k} \sum_{j=0}^{n} \binom{i+j}{i} x^i y^j, \tag{18}$$

$$J_2(n,x,y) = \lim_{k\to\infty} \sum_{i=0}^{k} \sum_{j=0}^{n} \binom{i+j}{i} i\, x^i y^j, \tag{19}$$

$$J_3(n,x,y) = \lim_{k\to\infty} \sum_{i=0}^{k} \sum_{j=0}^{n} \binom{i+j}{i} j\, x^i y^j. \tag{20}$$

Let $0 \le x < 1$ and $y \ge 0$. We can rewrite (18) as

$$\lim_{k\to\infty} \sum_{j=0}^{n} \frac{y^j}{j!} \sum_{i=0}^{k} \frac{(i+j)!}{i!} x^i$$

$$= \lim_{k\to\infty} \sum_{j=0}^{n} \frac{y^j}{j!} \frac{d^j}{dx^j} \sum_{i=0}^{k} x^{i+j}$$

$$= \sum_{j=0}^{n} \frac{y^j}{j!} \frac{d^j}{dx^j} \sum_{i=0}^{\infty} x^{i+j}$$

$$= \sum_{j=0}^{n} \frac{y^j}{j!} \frac{d^j}{dx^j} \left[\frac{x^j}{1-x}\right]$$

$$= \sum_{j=0}^{n} \frac{y^j}{j!} \sum_{k=0}^{j} \binom{j}{k} \left[\frac{d^k}{dx^k} x^j\right] \left[\frac{d^{j-k}}{dx^{j-k}} \frac{1}{1-x}\right]$$

$$= J_1(n,x,y).$$

The interchange of the differential operator and the infinite sum is allowed since $x < 1$. Analogously we get for $0 \le x < 1$, $y \ge 0$ from (19)

$$\lim_{k\to\infty} \sum_{j=0}^{n} \frac{y^j}{j!} \sum_{i=1}^{k} \frac{(i+j)!}{(i-1)!} x^i$$

$$= \sum_{j=0}^{n} \frac{x\, y^j}{j!} \frac{d^{j+1}}{dx^{j+1}} \left[\frac{x^{j+1}}{1-x}\right]$$

$$= J_2(n,x,y).$$

Finally, for $0 \le x < 1$, $y \ge 0$, we can derive J_3 in a similar way as (18), since it equals

$$\sum_{j=1}^{n} \frac{y^j}{(j-1)!} \sum_{i=0}^{\infty} \frac{(i+j)!}{i!} x^i. \qquad \square$$

For the performance measures we can derive some intuitively appealing inequalities as is shown by the following lemma.

LEMMA 3.2 (MONOTONICITY PROPERTIES)
For all M_1, $M_2 \in \mathbb{N}$ *and* $i = 1, 2$,

$$T_1(M_1, M_2) < T_1(M_1 + 1, M_2), \quad T_1(M_1, M_2) > T_1(M_1, M_2 + 1), \tag{21}$$

$$T_2(M_1, M_2) > T_2(M_1 + 1, M_2), \quad T_2(M_1, M_2) < T_2(M_1, M_2 + 1), \tag{22}$$

$$L_i(M_1, M_2) < L_i(M_1 + 1, M_2), \quad L_i(M_1, M_2) < L_i(M_1, M_2 + 1), \tag{23}$$

$$S_i(M_1, M_2) < S_i(M_1 + 1, M_2), \quad S_i(M_1, M_2) < S_i(M_1, M_2 + 1). \tag{24}$$

PROOF. See DE WAAL AND VAN DIJK [17].

Note that these inequalities are indeed what we would intuitively expect. For example (21) states that increasing the threshold M_1 for class 1 while keeping M_2 constant, gives a higher throughput T_1, since obviously more class 1 customers are going to be admitted to the queue. Conversely the throughput T_2 for class 2 will decrease as less processor capacity will be left to serve class 2 customers.

In DE WAAL AND VAN DIJK [17] the monotonicity properties of Lemma 3.2 are proven for a service discipline that is more general than standard Processor Sharing. This generalisation includes the model in COHEN [2], and also non-symmetric modifications. The monotonicity of throughputs and mean queue lengths is proven by treating these performance measures as average rewards and showing that monotonicity exists for finite horizon rewards. The inequality (24) for S_2 is an immediate result of (22) and (23) by application of Little's law (8). The inequality (24) for S_1 can, however, not be derived in this manner and has to be proven explicitly. This is the only monotonicity property where the product form equilibrium distribution is actually needed.

The monotonicity results of Lemma 3.2 remain valid when one of the thresholds becomes infinite, though for obvious reasons the inequality signs $<$ and $>$ have to be replaced by $\leq$ and $\geq$, respectively.

3.2 Existence of an optimal partition

We shall now use the monotonicity results of the preceding section to give conditions for the existence of a solution to problem (P_P). The first lemma shows that the existence of a feasible partition is sufficient for the existence of an optimal partition. This result is not trivial, since the existence of a feasible partition does not guarantee that there is a partition that attains the maximum class 1 throughput.

LEMMA 3.3 *There exists a feasible policy for problem* (P_P)*, if and only if there exists an optimal policy.*

PROOF. Since an optimal partition is by definition feasible, the "only if" statement is immediate. The proof for the "if" part proceeds by contradiction. Assume that there is a feasible partition, but no optimal partition. Since T_1 is bounded by λ_1, this means that there is no feasible partition that attains a maximum class 1 throughput and thus there must be infinitely many feasible partitions. Consequently there exists a sequence $\{C_n\}_{n=0}^{\infty}$ of feasible partitions satisfying

(a.i) $T_1(\mathcal{C}_n)$ is strictly increasing for $n \in \mathbb{N}$.

(a.ii) For any feasible partition $\mathcal{C}$ there exists an $n_{\mathcal{C}} \in \mathbb{N}$ such that $T_1(\mathcal{C}_n) > T_1(\mathcal{C})$ for $n > n_{\mathcal{C}}$.

In the remainder of this proof we shall use the notation $M_i(\mathcal{C})$ for $i = 1, 2$, to denote the class i threshold of partition $\mathcal{C}$. Note that $M_i(\mathcal{C})$ can be infinite.

First we shall prove that

$$\sup_{n\in\mathbb{N}} M_1(\mathcal{C}_n) = \infty. \tag{25}$$

Assume that (25) is not true, so $M_1(\mathcal{C}_n)$ takes only a finite number of discrete values. Consequently there must exist a subsequence $\{\mathcal{C}_n^{(1)}\}_{n=0}^{\infty}$ of $\{\mathcal{C}_n\}_{n=0}^{\infty}$, with $M_1(\mathcal{C}_n^{(1)})$ constant and equal to $\limsup_{n\to\infty} M_1(\mathcal{C}_n)$ which we shall denote as $\overline{M}_1$. Since $T_1(\mathcal{C}_n)$ and thus also $T_1(\mathcal{C}_n^{(1)})$ is strictly increasing, $M_2(\mathcal{C}_n^{(1)})$ must take infinitely many different values due to (21). Therefore, we can construct a subsequence $\{\mathcal{C}_n^{(2)}\}_{n=0}^{\infty}$ of $\{\mathcal{C}_n^{(1)}\}_{n=0}^{\infty}$ such that

(b.i) $M_1(\mathcal{C}_n^{(2)})$ is constant (by definition of $\mathcal{C}_n^{(1)}$ and thus also of $\mathcal{C}_n^{(2)}$).

(b.ii) $M_2(\mathcal{C}_n^{(2)})$ is strictly increasing in $n \in \mathbb{N}$.

(b.iii) $T_1(\mathcal{C}_n^{(2)})$ is strictly increasing in $n \in \mathbb{N}$.

This is in contradiction with the monotonicity property (21) of Lemma 3.2, however. We may thus conclude that $\sup_{n\in\mathbb{N}} M_1(\mathcal{C}_n) = \infty$.

If we have a feasible partition $\mathcal{C}_n$ for some $n \in \mathbb{N}$ with $M_1(\mathcal{C}_n) = \infty$, then $T_1(\mathcal{C}_n) = \lambda_1$ by Lemma 3.1, and this contradicts (a.ii). We may therefore assume that for all $n \in \mathbb{N}$, $M_1(\mathcal{C}_n) < \infty$. Since $\sup_{n\in\mathbb{N}} M_1(\mathcal{C}_n) = \infty$, there must exist a subsequence $\{\mathcal{C}_n^{(3)}\}_{n=0}^{\infty}$ of $\{\mathcal{C}_n\}_{n=0}^{\infty}$ with $M_1(\mathcal{C}_n^{(3)}) \uparrow \infty$ as $n \to \infty$. With regard to the sequence $\{M_2(\mathcal{C}_n^{(3)})\}_{n=0}^{\infty}$ we can distinguish between the following two cases:

(c.i) $M_2(\mathcal{C}_n^{(3)})$ takes only finitely many values for $n \in \mathbb{N}$.

(c.ii) $M_2(\mathcal{C}_n^{(3)})$ takes infinitely many values for $n \in \mathbb{N}$.

If (c.i) holds, then we can construct (yet another) subsequence $\{\mathcal{C}_n^{(4)}\}_{n=0}^{\infty}$ of $\{\mathcal{C}_n^{(3)}\}_{n=0}^{\infty}$, such that $M_2(\mathcal{C}_n^{(4)})$ is constant and equal to $\limsup_{n\to\infty} M_2(\mathcal{C}_n^{(3)})$ which we shall denote as $\overline{M}_2$. For all $n \in \mathbb{N}$ we then have by the definition of feasibility, $S_1(M_1(\mathcal{C}_n^{(4)}), \overline{M}_2)$ increasing and bounded from above by S, and $T_2(M_1(\mathcal{C}_n^{(4)}), \overline{M}_2)$ decreasing and bounded from below by T. Furthermore, by definition of $\{\mathcal{C}_n^{(3)}\}_{n=0}^{\infty}$, we have $M_1(\mathcal{C}_n^{(4)}) \uparrow \infty$ as $n \to \infty$ and thus $\langle \infty, \overline{M}_2 \rangle$ is a feasible partition. Since $T_1(\infty, \overline{M}_2) = \lambda_1$, this leads to a contradiction with (a.ii).

Finally in case (c.ii) it is possible to construct a subsequence $\{\mathcal{C}_n^{(5)}\}_{n=0}^{\infty}$ of $\{\mathcal{C}_n^{(3)}\}_{n=0}^{\infty}$ with $M_2(\mathcal{C}_n^{(5)})$ strictly increasing to ∞ for $n \in \mathbb{N}$. Since by construction $M_1(\mathcal{C}_n^{(3)})$ and thus also $M_1(\mathcal{C}_n^{(5)})$ is strictly increasing, we may conclude that $\langle \infty, \infty \rangle$ is a feasible partition. Since $T_1(\infty, \infty) = \lambda_1$, we get again a contradiction with (a.ii). As a result we may thus conclude that there exists an optimal partition. □

With this lemma we can now proceed with sufficient conditions for the existence of feasible partitions. The first sufficient condition for existence concerns a queueing model where the parameters ρ_1 and ρ_2 and the bound S are chosen such that the constraint on S_1 is satisfied by all partitioning policies.

LEMMA 3.4 *If the queue is ergodic for the partitioning policy with thresholds* $M_1 = M_2 = \infty$, *i.e.* $\rho_1 + \rho_2 < 1$, *and if*

$$S \geq \frac{1}{\mu_1(1-\rho_1-\rho_2)} \tag{26}$$

then $M_1 = M_2 = \infty$ *is the optimal partitioning policy.*

PROOF. Note that if $\rho_1+\rho_2 < 1$, then the right hand side of (26) is the expression for $S_1(\infty,\infty)$ (cf. REISER AND KOBAYASHI [13, equation (37)]). If the inequality (26) holds, then we know from Lemma 3.2 that $S_1(M_1,M_2) \leq S$ for all M_1, $M_2 \in \mathbb{N}$. Furthermore $T_i(M_1,M_2) \leq \lambda_i$ for M_1, $M_2 \in \mathbb{N}$ and $i = 1, 2$, and this bound is tight for $M_1 = M_2 = \infty$. Therefore $\langle\infty,\infty\rangle$ is a feasible policy, since $T_2(\infty,\infty) \geq T$. The class 1 throughput $T_1(\infty,\infty) = \lambda_1$, so $\langle\infty,\infty\rangle$ is the optimal partition. □

The following two lemmas give necessary and sufficient conditions for less trivial parameter settings.

LEMMA 3.5 *If the lower bound* $T = \lambda_2$, *then there exists an optimal partition if and only if* $S_1(1,\infty) \leq S$.

PROOF. Observe that from (5) the throughput $T_2(C)$ of class 2 under policy f_C is equal to the arrival rate λ_2 times the probability that a class 2 customer is admitted under f_C. Since the blocking probability for class 2 is zero only for partitions with threshold $M_2 = \infty$, a feasible policy must thus use an infinite class 2 threshold. If $S_1(1,\infty) > S$, then by the inequality (24) we have $S_1(M_1,\infty) > S$ for all $M_1 \in \mathbb{N}$ and thus no feasible partition exists. If $S_1(1,\infty) \leq S$, then there exists at least one feasible partition, viz. $\langle 1,\infty\rangle$, and thus there exists an optimal partition. □

LEMMA 3.6 *If* $T < \lambda_2$, *then there exists an optimal partition if and only if* $S \geq S_1(1,K)$, *where* K *is defined as*

$$K = \min\{\, M_2 \in \mathbb{N} \mid T_2(1,M_2) \geq T \,\}. \tag{27}$$

PROOF. Let $T < \lambda_2$ and let K be defined as in (27). Such a K exists and is finite since $T_2(1,M_2) \uparrow \lambda_2$ as $M_2 \to \infty$. From the monotonicity of T_2 we know that a partition $\langle M_1,M_2\rangle$ can be feasible only if M_2 is at least K. If $S_1(1,K) \leq S$, then $\langle 1,K\rangle$ is a feasible partition and thus an optimal policy exists. If $S_1(1,K) > S$, then by (24) for all $M_1 \geq 1$ and $M_2 \geq K$ we have $S_1(M_1,M_2) > S$, and consequently there are no feasible policies. □

We can derive closed form expressions for $S_1(1,M_2)$ and $T_2(1,M_2)$, $M_2 \in \mathbb{N}$, by employing the relations (4)–(8). If we define for $n \in \mathbb{N}$

$$J_4(n) = \sum_{j=0}^{n} \rho_2^j = \frac{1-\rho_2^{n+1}}{1-\rho_2}, \tag{28}$$

and

$$J_5(n) = \sum_{j=0}^{n} (j+1)\rho_1\rho_2^j = \rho_1 \frac{1-(n+2)\rho_2^{n+1}+(n+1)\rho_2^{n+2}}{(1-\rho_2)^2}, \tag{29}$$

one can show by (5)–(8) that

$$S_1(1, M_2) = \frac{J_5(M_2)}{\lambda_1(J_4(M_2) + J_5(M_2))} \tag{30}$$

and

$$T_2(1, M_2) = \lambda_2 \frac{J_4(M_2 - 1) + J_5(M_2 - 1)}{J_4(M_2) + J_5(M_2)}. \tag{31}$$

The expressions (30) and (31) can be evaluated by a recursion in n for $J_4(n)$ and $J_5(n)$. The monotonicity of S_1 and T_2 can be employed to search for a K satisfying (27).

3.3 Design of the optimal partition

Once the existence of a feasible partition has been established, we can use the monotonicity properties of Lemma 3.2 to design a procedure to find the optimal partition. Assume for instance that we have a partition $\langle M_1, M_2 \rangle$ that violates the constraint $S_1(M_1, M_2) \leq S$. From the monotonicity of S_1 we then know that a feasible partition $\langle M_1^*, M_2^* \rangle$ must satisfy $M_i^* \leq M_i$, $i = 1, 2$.

Since we have expressions for all the performance measures that are involved in the formulation of problem (P_P), we are all set to discuss the algorithm to find the optimal partition. In this procedure we can distinguish the following steps.

(i) Test if there exists a feasible partition.

(ii) Determine the set of feasible partitions.

(iii) Restrict the set of feasible partitions to a finite set that contains the optimal partition.

(iv) Find the optimal partition over this finite set.

It is not necessary in all cases to perform all of these four steps, since for some examples we can already conclude in steps (i) or (ii) what the optimal partition is (if any).

For step (i) we use the results that were obtained in Lemma's 3.4–3.6. When we have a situation where Lemma 3.4 applies, then obviously we can stop, since we have found an optimal partition. If this is not the case, then we can check whether we have an example of Lemma 3.5. If this is true, then we know from Lemma 3.5 that the optimal partition must be of the form $\langle M_1, \infty \rangle$ for some $M_1 \in \mathbb{N}$. Note that the value of $S_1(M_1, \infty)$ can be computed from Lemma 3.1. If there exists a feasible partition, then clearly the optimal partition is $\langle M_1^*, \infty \rangle$, with

$$M_1^* = \max\{ M_1 \in \mathbb{N} \mid S_1(M_1, \infty) \leq S \}.$$

Such an M_1^* exists and is finite, since $S_1(M_1, \infty) \uparrow \infty$ as $M_1 \to \infty$ (otherwise the conditions of Lemma 3.4 would have been satisfied).

If we can exclude the cases where Lemma 3.4 and 3.5 apply, then we are in a situation where at least one of the two thresholds of the optimal partition must be finite. We can then characterise the set of feasible policies. We do this by determining for any fixed value of M_2 the maximally feasible class 1 threshold. We define for all $M_2 \in \mathbb{N} \cup \{\infty\}$

$$\mathcal{M}_1^T(M_2) = \sup\{\, M_1 \in \mathbb{N} \mid T_2(M_1, M_2) \geq T \,\},$$

$$\mathcal{M}_1^S(M_2) = \sup\{\, M_1 \in \mathbb{N} \mid S_1(M_1, M_2) \leq S \,\},$$

$$\mathcal{M}_1(M_2) = \min\{\, \mathcal{M}_1^T(M_2)\,,\, \mathcal{M}_1^S(M_2) \,\}.$$

For any fixed M_2 a partition $\langle M_1, M_2 \rangle$ can be feasible only if $M_1 \leq \mathcal{M}_1(M_2)$. An immediate result from the monotonicity properties is the local optimality of the class 1 threshold $\mathcal{M}_1(M_2)$ for fixed M_2, since for that threshold the maximal class 1 throughput is obtained.

The set of all feasible partitions can now be characterised as

$$\{\, \langle M_1, M_2 \rangle \mid M_2 \in \mathbb{N} \cup \{\infty\},\, 0 \leq M_1 \leq \mathcal{M}_1(M_2) \,\}.$$

and for optimisation purposes this set can be restricted to

$$\{\, \langle \mathcal{M}_1(M_2), M_2 \rangle \mid M_2 \in \mathbb{N} \cup \{\infty\} \,\},$$

that contains the optimal partition. Examination of the structure of this set reveals that there are two difficulties in the evaluation of T_1 over this set. First we see that we have infinitely many values for M_2 and second we can encounter the situation where $\mathcal{M}_1(M_2) = \infty$ for some M_2. With respect to the latter possibility we can remark that this is in fact no difficulty, since if for some M_2 the partition $\langle \infty, M_2 \rangle$ is feasible, then it is also optimal since $T_1(\infty, M_2) = \lambda_1$.

The remaining problem thus concerns the infinitely many values that M_2 can take. We shall show that we can restrict ourselves to a finite set, thus reducing the search of the optimal partition to a search over a finite set. For this we need the following lemma.

LEMMA 3.7

(i) $\mathcal{M}_1^T(m)$ *is non-decreasing in* m.

(ii) $\mathcal{M}_1^S(m)$ *is non-increasing in* m.

(iii) $\mathcal{M}_1(m)$ *is unimodal in* m, *i.e.* $\mathcal{M}_1(m-1) \geq \mathcal{M}_1(m)$ *implies* $\mathcal{M}_1(m) \geq \mathcal{M}_1(m+1)$, *and* $\mathcal{M}_1(m) \leq \mathcal{M}_1(m+1)$ *implies* $\mathcal{M}_1(m-1) \leq \mathcal{M}_1(m)$.

PROOF. Parts (i) and (ii) are immediate from the monotonicity of T_2 and S_1, respectively. Part (iii) follows from the definition of $\mathcal{M}_1$ and (i) and (ii). □

Recall from the previous steps of the algorithm, that we can exclude the possibility of a feasible partition with two infinite thresholds. We may therefore conclude by the unimodality of $\mathcal{M}_1$ that $\mathcal{M}_1(m)$ converges to a finite value $\mathcal{M}_1(\infty)$ as $m \to \infty$. This limit can be computed from the expressions for $S_1(M_1, \infty)$ and $T_2(M_1, \infty)$. Since $\mathcal{M}(m)$ can thus take only finitely many values in $\mathbb{N} \cup \{\infty\}$, there must be a finite $\mathcal{M}_2$ such that $\mathcal{M}_1(M_2) = \mathcal{M}_1(\infty)$ for $M_2 \geq \mathcal{M}_2$. From the monotonicity of T_1 it is clear that within the set

$$\{\, \langle \mathcal{M}_1(M_2), M_2 \rangle \mid M_2 \geq \mathcal{M}_2 \,\}$$

the maximum class 1 throughput is obtained for partition $\langle \mathcal{M}_1(\mathcal{M}_2), \mathcal{M}_2 \rangle$. Therefore it is sufficient to restrict the search for the optimal partition to the finite set

$$\{\langle \mathcal{M}_1(M_2), M_2\rangle \mid 0 \le M_2 \le \mathcal{M}_2\}.$$

Summing up the algorithm we thus have:

ALGORITHM 3.8

(i) *If* $\rho_1 + \rho_2 < 1$ *and* $S \ge (\mu_1(1-\rho_1-\rho_2))^{-1}$, *then the optimal partition is* $\langle\infty,\infty\rangle$. *Otherwise continue with step (ii).*

(ii) *If* $T = \lambda_2$ *and* $S_1(1,\infty) > S$, *then no feasible policy exists. If* $T = \lambda_2$ *and* $S_1(1,\infty) \le S$, *then the optimal policy is* $\langle M_1^*, \infty\rangle$, *where* M_1^* *can be computed with the aid of Lemma 3.1 as*

$$M_1^* = \max\{ M_1 \in \mathbb{N} \mid S_1(M_1,\infty) \le S\}.$$

If the above conditions do not apply, then proceed with step (iii).

(iii) *Compute with (31) the value of* K *that satisfies*

$$K = \min\{ M_2 \in \mathbb{N} \mid T_2(1,M_1) \ge T\}.$$

Compute $S_1(1,K)$ *from (30). If* $S_1(1,K) > S$, *then no feasible partition exists. Otherwise continue with Algorithm 3.9.*

ALGORITHM 3.9
Define $\mathcal{M}_1 : \mathbb{N}\cup\{\infty\} \to \mathbb{N}\cup\{\infty\}$ *as*

$$\mathcal{M}_1(M_2) = \sup\{ M_1 \mid S_1(M_1,M_2) \le S\,,\, T_2(M_1,M_2) \ge T\}.$$

(i) *Compute* $\mathcal{M}_1(\infty)$ *with the aid of Lemma 3.1, and compute* $\mathcal{M}_2$, *that is defined as*

$$\mathcal{M}_2 = \min\{ M_2 \mid \mathcal{M}_1(M_2) = \mathcal{M}_1(\infty)\}.$$

(ii) *Let* $M_2 = 0$.

(iii) *Compute* $\mathcal{M}_1(M_2)$ *and* $T_1(\mathcal{M}_1(M_2),M_2)$. *If for some* M_2 *we get* $\mathcal{M}_1(M_2) = \infty$, *then* $\langle\infty, M_2\rangle$ *is an optimal partition. If* $M_2 = \mathcal{M}_2$, *then proceed with step (iv). Otherwise increase* M_2 *by 1 and repeat step (iii).*

(iv) *Determine the value of* M_2, $0 \le M_2 \le \mathcal{M}_2$ *that attains the maximum value for* $T_1(\mathcal{M}_1(M_2),M_2)$.

4 SHARING POLICIES

In this section we shall briefly discuss sharing policies. We know from LAM [8] and DE WAAL AND VAN DIJK [16] that for sharing policies the equilibrium distribution is given by the product form (4). This means that the performance measures can be evaluated quite easily. The main part of the algorithm is the computation of the normalisation constant $G(C)$. We shall not discuss the evaluation algorithm here, but refer to KAUFMAN [5, Section V].

In contrast with the partitioning policies, there does not exist monotonicity of the performance measures with respect to the weight factors c_1, c_2, and the poolsize c. This can be proven explicitly for some simple sharing policies. Consider for example a sharing policy with weight factors $c_1 = 2$ and $c_2 = 3$. If the poolsize $c = 2$, then the admission region consists of the states $(0,0)$ and $(1,0)$. This means that according to (1) a class 1 customer will be admitted to the queue only if it is empty, so the class 1 throughput in this case is

$$\lambda_1 \frac{\bar{\pi}(0,0)}{\bar{\pi}(0,0)+\bar{\pi}(1,0)} = \frac{\lambda_1}{1+\rho_1}.$$

Increasing the poolsize to $c = 3$ adds the state $(0,1)$ to the admission region. The effect is that class 1 customers are still admitted only when they arrive to an empty queue. The class 1 throughput for $c = 3$ is thus

$$\lambda_1 \frac{\bar{\pi}(0,0)}{\bar{\pi}(0,0)+\bar{\pi}(1,0)+\bar{\pi}(0,1)} = \frac{\lambda_1}{1+\rho_1+\rho_2},$$

which is smaller than the throughput for $c = 2$. If we increase the poolsize to $c = 4$, then the state (2,0) is added to the admission region. Now class 1 customers will also be admitted to the queue if the state is (1,0) at the moment of arrival. The class 1 throughput for $c = 4$ is thus

$$\lambda_1 \frac{\bar{\pi}(0,0)+\bar{\pi}(1,0)}{\bar{\pi}(0,0)+\bar{\pi}(1,0)+\bar{\pi}(0,1)+\bar{\pi}(2,0)} = \lambda_1 \frac{1+\rho_1}{1+\rho_1+\rho_2+\rho_1^2},$$

which is larger than T_1 for $c = 3$.

The lack of monotonicity for less trivial examples can be illustrated by Figures 5–7 where the throughputs T_1, T_2, and the expected sojourn time S_1 of a sharing policy are depicted as a function of the poolsize. The system under consideration is a queue with parameters $\lambda_1 = \lambda_2 = 0.5$, $\mu_1 = \mu_2 = 1.0$, and weight factors $c_1 = 5$, $c_2 = 9$. In fact the only performance measure that exhibits monotonicity is the expected class 1 sojourn time S_1. This should not come as a surprise, since an increase of the poolsize will always lead to more customers being admitted and a decrease of the processor's idle time. We have not been able, however, to prove this monotonicity property.

With regard to the throughputs T_1 and T_2, one can observe in Figures 5 and 6 that T_i, $i = 1, 2$, is monotone increasing if the poolsize is increased by a multiple of the weight factor c_1. This observation and the fact that T_i does not necessarily increase when the poolsize is increased by one, can partly be explained from Figure 8. Compare for example the admission regions for $c = 17$ and $c = 18$. The regions differ only in one state, viz. $\mathbf{m} = (0,2)$, and this extra state is beneficial only for class 2 customers, since it does not change the admission policy for class 1. Consequently class 2 throughput is larger for $c = 18$ than for $c = 17$, while the opposite is true for T_1. A reverse effect can be observed in increasing c from 19 to 20. Note also that for $c = 20, 21, 22$, the admission regions are identical and so are the performance measures.

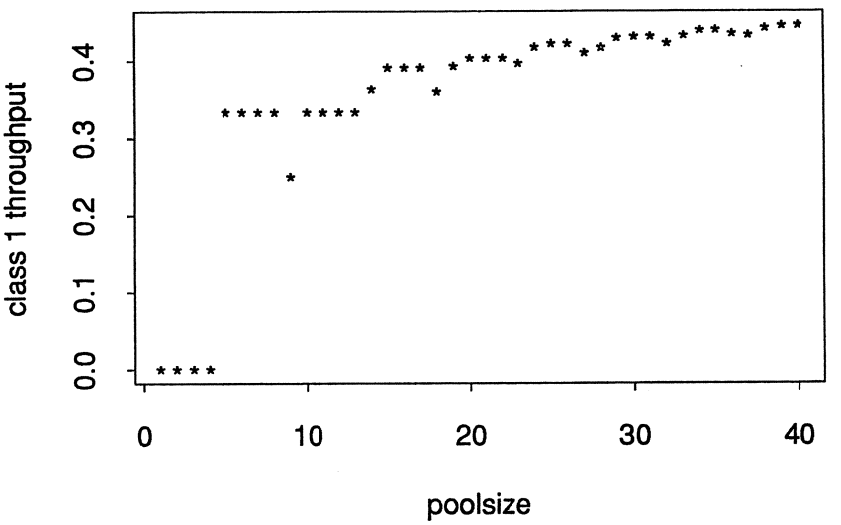

FIGURE 5. Throughput of class 1 as a function of the poolsize. Parameters $\lambda_1 = \lambda_2 = 0.5$, $\mu_1 = \mu_2 = 1.0$, weight factors $c_1 = 5$, $c_2 = 9$.

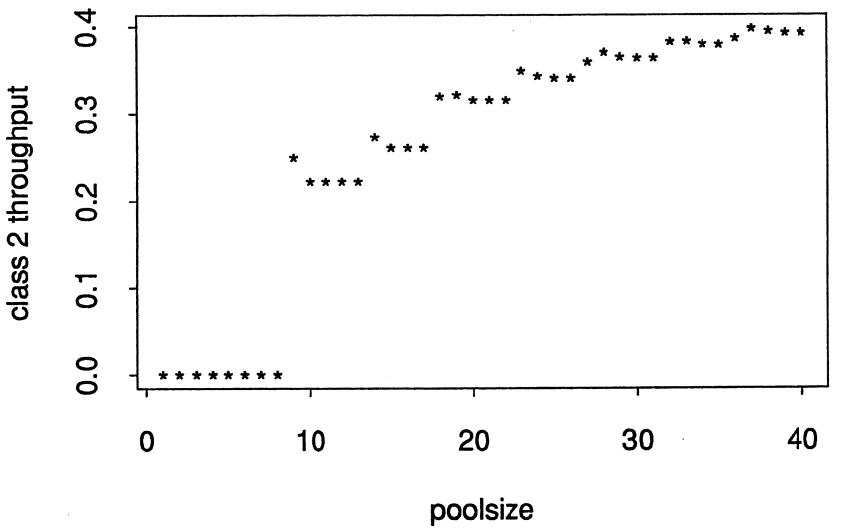

FIGURE 6. Throughput of class 2 as a function of the poolsize. Parameters $\lambda_1 = \lambda_2 = 0.5$, $\mu_1 = \mu_2 = 1.0$, weight factors $c_1 = 5$, $c_2 = 9$.

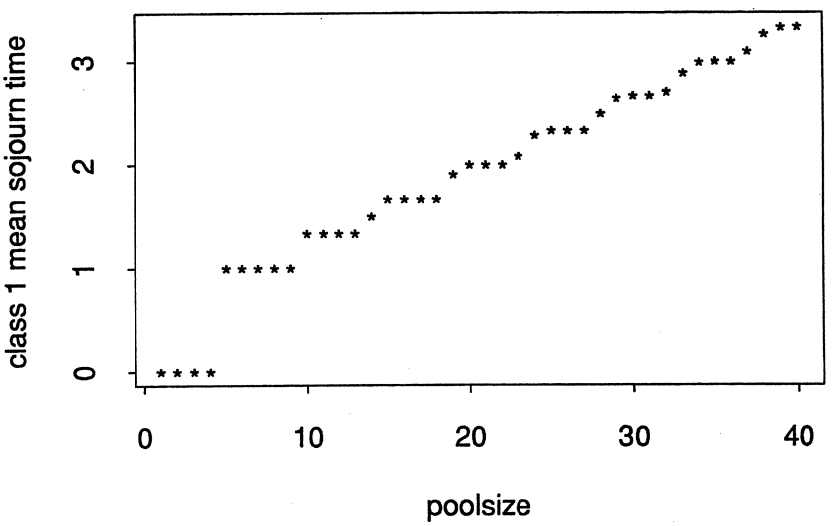

FIGURE 7. Mean sojourn time of class 1 as a function of the poolsize. Parameters $\lambda_1 = \lambda_2 = 0.5, \mu_1 = \mu_2 = 1.0$, weight factors $c_1 = 5, c_2 = 9$.

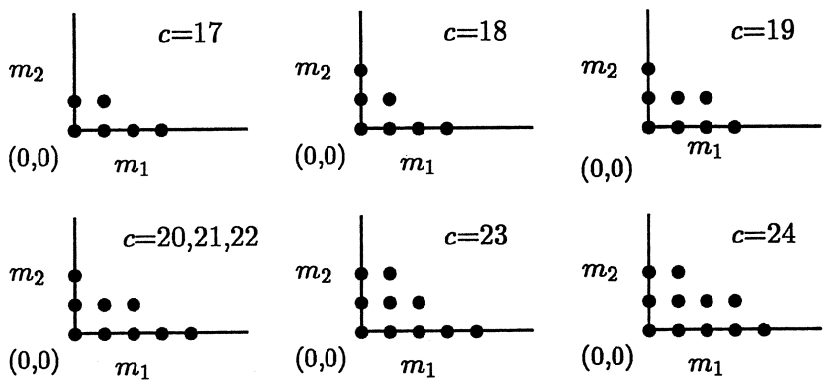

FIGURE 8. Admission region as a function of the poolsize c. Weight factors $c_1 = 5$, $c_2 = 9$.

5 NUMERICAL RESULTS

In this section we shall discuss the performance of partitioning and sharing policies from numerical examples. For several parameter settings of a queueing model the optimal partitions are computed and the robustness of this class of policies is discussed. For the same set of parameters we compute sharing policies of which the performance is close to the corresponding optimal partitioning policies. The robustness of sharing policies is also included in the exposition.

The queueing system we consider has class 2 arrival rate $\lambda_2 = 0.3$, and service rates $\mu_1 = \mu_2 = 1.0$. The lower bound on the class 2 throughput is set at $T = 0.2$ and the upper bound on the mean class 1 sojourn time is $S = 20$. In Table 1 the values of the optimal partitions' thresholds are presented for different values of the class 1 arrival rate λ_1. Included in the table are the values of the performance measures for these partitions. The third column in the table refers to the so-called *normalised arrival rate* $\bar{\lambda}_1$. Due to the lower bound $T = 0.2$ on the class 2 throughput, only a fraction 0.8 of the processor's speed is available to serve class 1 customers. Since $\mu_1 = 1.0$, this means that $\lambda_1 = 0.8$ corresponds to a call request load of 100%. For this reason the normalised arrival rate is introduced, and it is defined as $\bar{\lambda}_1 = \lambda_1/0.8$.

From Lemma 3.4 we know that for $\lambda_1 \leq 0.65$ ($\bar{\lambda}_1 \leq 0.81$) the mean sojourn time of class 1 will always be smaller than 20, and thus partition $\langle \infty, \infty \rangle$ is feasible and optimal. For $\lambda_1 \leq 0.80$ ($\bar{\lambda}_1 \leq 1.00$) the optimal class 1 threshold is infinite. We have not included these policies in the results, since they are uninteresting when we come to the robustness issue. Because of the infinite threshold, the queue will no longer be ergodic when $\bar{\lambda}_1 \geq 1.00$.

In Figures 9–11 the performance measures for the six partitions of Table 1 are depicted when the class 1 arrival rate is varied while the thresholds are kept constant. The numbers in the figures correspond to the numbers of the partitions in Table 1. From Figure 9 we can conclude that the largest values of T_1 can be obtained by using partition No. 1 — optimal for $\bar{\lambda}_1 = 1.0$ — for all arrival rates. This observation can be explained immediately from the monotonicity properties of Lemma 3.2. Due to the large thresholds of this partition the bounds on T_2 and S_1 are violated for large $\bar{\lambda}_1$, however. Conversely, partition No. 6 that is optimal for $\bar{\lambda}_1 = 1.5$ satisfies the constraints for all values of $\bar{\lambda}_1 \in [0.0, 4.0]$ but gives for $\bar{\lambda}_1 = 1.0$ a class 1 throughput that is about 7% less than the optimal value at that load. If one considers this loss to be too large, then clearly partitioning policies are not robust.

Next we discuss the performance of sharing policies. We have considered the processor sharing queueing model for the same parameter settings as the partitioning policies. From Section 4 we know that sharing policies do not exhibit monotonicity, that can be used to search for optimal policies. The performance measures, however, can be computed quite efficiently, thus allowing an "interactive" search for satisfying policies, i.e. policies of which the performance is close to the optimal partitioning policies. In the examples we found that the following rules can be applied in the search for a satisfying sharing policy if $\bar{\lambda}_1 \geq 1.0$. First we have to search for the right ratio of the weight factors to get the throughputs T_1 and T_2 close to 0.8 and 0.2, respectively. Having found the right ratio, we then have to vary the poolsize c until S_1 is close to 20. In most cases we were able to find sharing policies of which the performance was close to that of partitioning policies (see Table 2). From this table we can conclude that when $\bar{\lambda}_1$ increases, then the ratio of c_1 and c_2 has to shift to give class 2 a preference over class 1 (compare policies No. 1 and No. 6).

No.	λ_1	$\bar{\lambda}_1$	M_1^{opt}	M_2^{opt}	T_1	T_2	S_1
1	0.80	1.00	32	5	0.788	0.201	19.7
2	0.85	1.06	23	5	0.792	0.202	18.8
3	0.90	1.13	20	5	0.794	0.203	18.2
4	1.00	1.25	18	5	0.798	0.201	18.2
5	1.20	1.50	16	5	0.797	0.203	17.7
6	1.50	1.88	15	5	0.796	0.204	17.5

TABLE 1. Optimal partitioning policies.

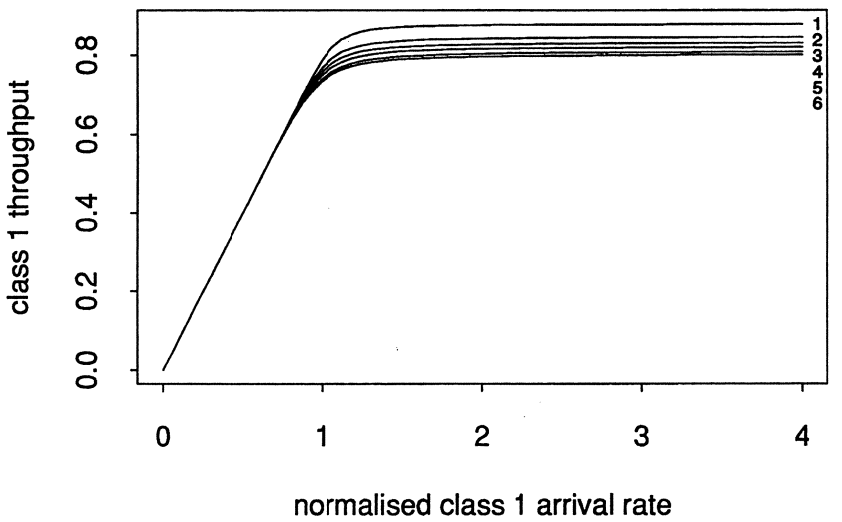

FIGURE 9. Throughput of class 1 for partitioning policies.

No.	λ_1	$\bar{\lambda}_1$	c_1	c_2	c	T_1	T_2	S_1
1	0.80	1.00	1	15	80	0.781	0.200	15.2
2	0.85	1.06	1	6	50	0.795	0.200	19.8
3	0.90	1.13	1	3	34	0.793	0.205	19.7
4	1.00	1.25	4	7	105	0.798	0.201	19.3
5	1.20	1.50	12	11	263	0.799	0.201	20.0
6	1.50	1.88	5	3	99	0.797	0.203	19.9

TABLE 2. Satisfying sharing policies.

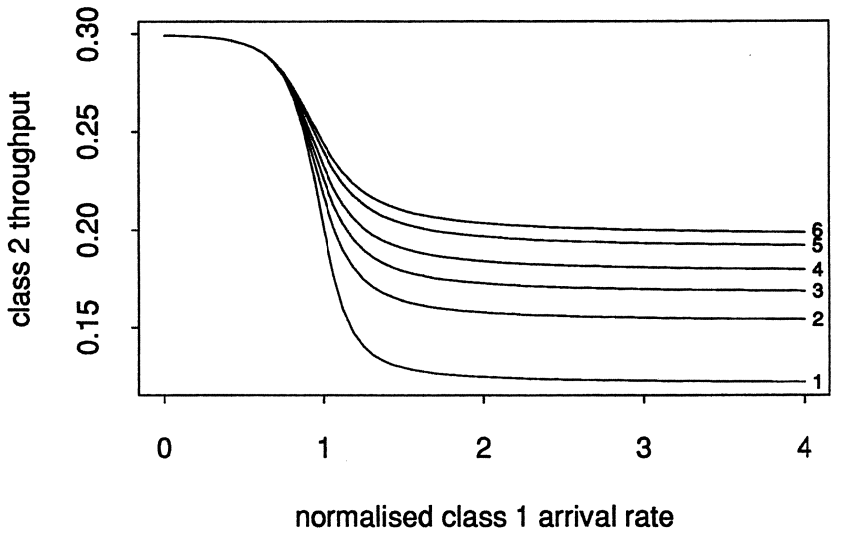

FIGURE 10. Throughput of class 2 for partitioning policies.

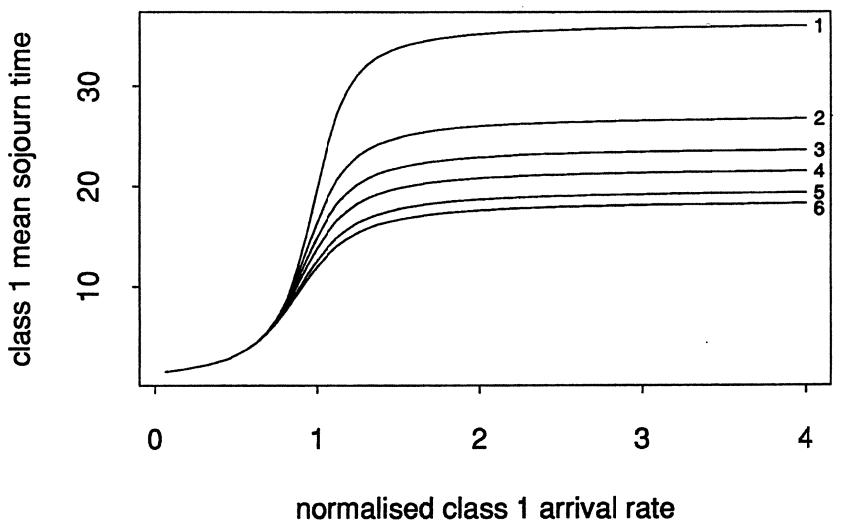

FIGURE 11. Mean sojourn time of class 1 for partitioning policies.

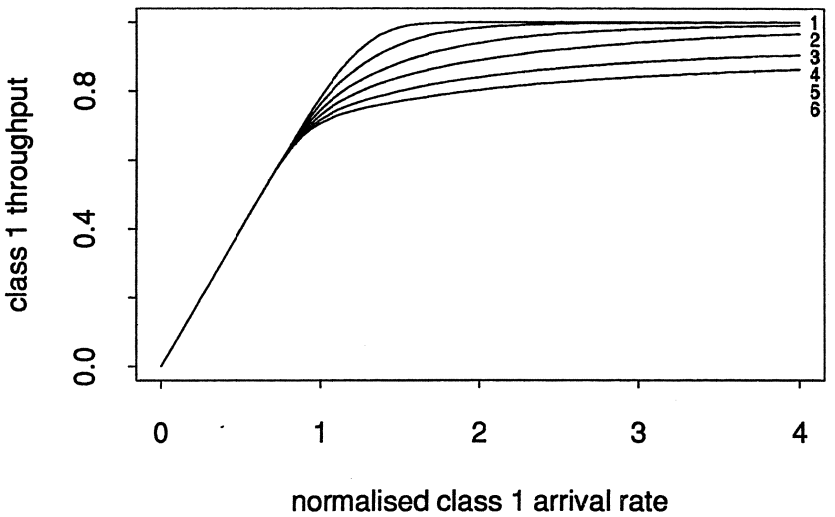

FIGURE 12. Throughput of class 1 for sharing policies.

Analogously to the investigation on partitioning policies we have examined the robustness of sharing policies. The results of these computations are shown in Figures 12–14. From these figures we can draw the same conclusions as for the partitioning policies, viz. the policy that is optimal for $\bar{\lambda}_1 = 1.0$ violates the constraints when the arrival intensity increases. In this case also the sharing policy that is optimal for $\bar{\lambda}_1 = 1.5$ is not feasible for $\bar{\lambda}_1 = 4.0$. The loss of class 1 throughput at $\bar{\lambda}_1 = 1.00$ of sharing policy No. 6 compared to No. 1 is 9%, so sharing policies are not robust either. For some sharing policies the class 2 throughput even decreases to zero when the load becomes high. This observation can be explained from Figure 8. If the weight factors are such that $c_1 > c_2$, then the admission region contains exactly one state with a maximal class 1 queue length. If the queue becomes overloaded, then the equilibrium distribution will have almost all of its mass in this state. Since this state has a zero class 2 queue length, hardly any class 2 customers will be admitted and T_2 becomes zero.

For both classes of admission policies we conjecture that the performance under varying loads can be improved by an adaptive control scheme. There are two approaches to construct adaptive partitioning or sharing policies. In the first approach we determine the optimal or satisfying policies for a number of values of λ_1. In the exchange the value of this parameter is estimated periodically and the policy is adjusted according to the latest estimate. This type of control is called *certainty equivalence adaptive control.*

In the second approach we use one or several system characteristics as indicators for the value of λ_1. As an example one can think of the queue length, since a large number of customers may indicate that the arrival rate is high. In this approach the admission policy should be adjusted periodically according to the actual queue length.

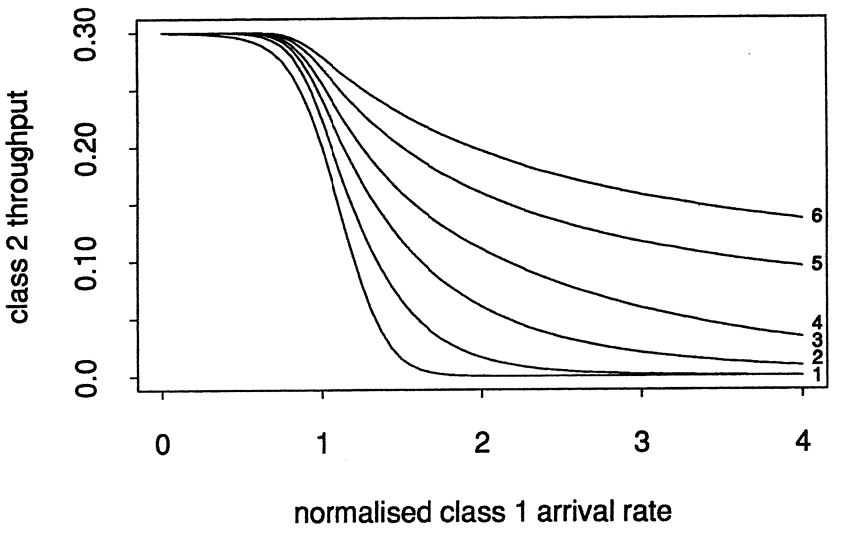

FIGURE 13. Throughput of class 2 for sharing policies.

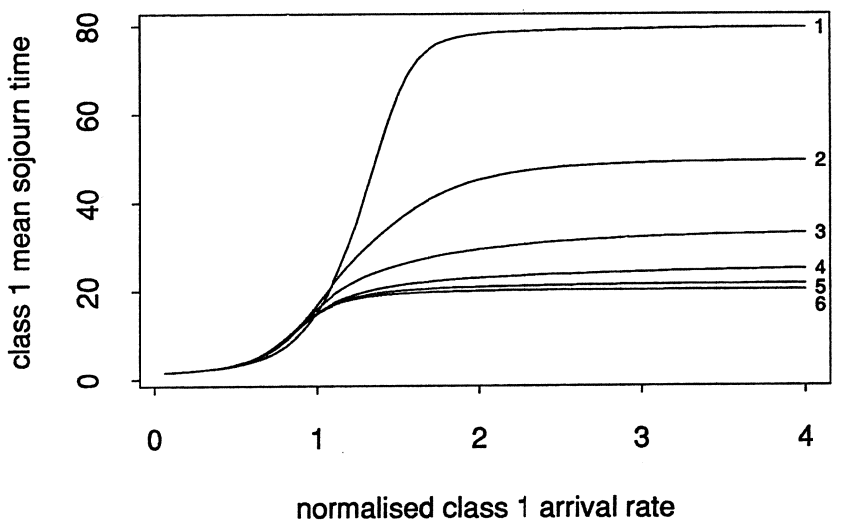

FIGURE 14. Mean sojourn time of class 1 for sharing policies.

REFERENCES

[1] J.L. VAN DEN BERG (1990). *Sojourn Times in Feedback and Processor Sharing Queues.* PhD thesis, Centre for Mathematics and Computer Science, Amsterdam.

[2] J.W. COHEN (1979). The Multiple Phase Service Network with Generalized Processor Sharing. *Acta Inform.* 12, 245–284.

[3] G.J. FOSCHINI AND B. GOPINATH (1983). Sharing Memory Optimally. *IEEE Trans. Comm.* COM-31, 352–359.

[4] A. HORDIJK AND F. SPIEKSMA (1989). Constrained Admission Control to a Queueing System. *Adv. Appl. Prob.* 21, 409–431.

[5] J.S. KAUFMAN (1981). Blocking in a Shared Resource Environment. *IEEE Trans. Comm.* COM-29, 1474–1481.

[6] F.P. KELLY (1979). *Reversibility and Stochastic Networks.* Wiley, New York.

[7] L. KLEINROCK (1967). Time-shared Systems: A Theoretical Treatment. *J. Assoc. Comput. Mach.* 14, 242–261.

[8] S.S. LAM (1977). Queueing Networks with Population Size Constraints. *IBM J. Res. Develop.* 21, 370–378.

[9] J.D. LITTLE (1961). A Proof of the Queueing Formula L=λW. *Oper. Res.* 22, 417–421.

[10] D.-J. MA AND A.M. MAKOWSKI (1987). Optimality Results for a Simple Flow Control Problem, in *Proc. of the 26th Conf. on Decision and Control*, 1852–1857, IEEE Press, Piscataway.

[11] P. NAIN AND K.W. ROSS. Optimal Multiplexing of Heterogeneous Traffic with Hard Constraint, in *Performance Evaluation Review*, 100–108.

[12] P. NAIN AND K.W. ROSS (1986). Optimal Priority Assignment with Hard Constraint. *IEEE Trans. Autom. Control* AC-31, 883–888.

[13] M. REISER AND H. KOBAYASHI (1975). Queueing Networks with Multiple Closed Chains: Theory and Computational Algorithms. *IBM J. Res. Develop.* 19, 283–294.

[14] T.G. ROBERTAZZI AND A.A. LAZAR (1985). On the Modeling and Optimal Flow Control of the Jacksonian Network. *Perf. Eval.* 5, 29–43.

[15] K.W. ROSS (1985). *Constrained Markov Decision Processes with Queueing Applications.* PhD thesis, Computer, Information and Control Engineering, University of Michigan.

[16] P.R. DE WAAL AND N.M. VAN DIJK (1989). Interconnected Networks of Queues with Randomized Arrival and Departure Blocking. Report BS-R8934, Centre for Mathematics and Computer Science, Amsterdam. (submitted).

[17] P.R. DE WAAL AND N.M. VAN DIJK (1989). Monotonicity of Performance Measures in a Processor Sharing Queue. Report BS-R8902, Centre for Mathematics and Computer Science, Amsterdam. (to appear in *Perf. Eval.*).